MASTER EN SCIENCES INFORMATIQUES

Adaptation du processus de tests logiciel en fonction du contexte: étude de cas

S'Jongers, Michaël

Award date:
2004

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année Académique 2003 - 2004

Adaptation du processus de tests logiciel en fonction du contexte

-

Etude de cas

Michaël S'Jongers

Mémoire présenté en vue de l'obtention du grade de Maître et Licencié en Informatique.

VTLS 2000191 5

Résumé

Les processus de développement et le langage de notation UML permettent de formaliser et de modéliser les différentes étapes du cycle de vie de développement logiciel.

Dans le cadre du projet SPINOV du CITI du CRP Henri Tudor, des travaux d'outillage méthodologique avec UML visent à améliorer la qualité des pratiques de développement logiciel.

L'objectif de mon mémoire est de proposer une démarche méthodologique de mise en oeuvre des tests logiciels, d'outiller la démarche d'un point de vue documentaire (modèle de document, check listes ...) et enfin d'étudier les possibilités offertes par UML de modéliser la phase de tests.

Abstract

Software development process and UML language allow to formalize and to modelize the differents stages of development process life cycle.

The SPINOV project of CITI is a methodology work with UML with intent to improve the quality of software development project.

The objective of this master thesis is the proposal of a testing approach, to equip the testing steps for documentation (templates, check lists ...) and to study the possibilities offered by UML for modeling the testing phase.

Remerciements :

Je désire remercier le Professeur Naji Habra pour son aide quant à ses conseils concernant la rédaction du mémoire.

Un merci tout particulier à Valérie Bétry pour le temps souvent précieux qu'elle m'a consacré pour répondre à mes nombreuses questions et à la relecture de mes rapports.

Merci également à Béatrix Barafort pour les documentations sur CMM, CMMI et SPICE.

Merci au CITI et particulièrement à l'équipe REF qui m'ont accueillis chaleureusement au sein de l'équipe.

Je remercie également toutes les personnes interviewées durant mon travail au centre.

Et merci à Ludo Houdenaert pour la relecture qu'il a effectué et à ses conseils.

Table des matières

Liste des tableaux

Table des figures

Introduction

J'ai effectué mon **stage** a Henri Tudor (CRP Henri Tudor) à Luxembourg au sein du Centre d'Innovation par les Technologies de l'Information (CITI). Le CITI m'a mis à disposition de l'équipe de référentiels de certification et de modélisation (REF), dont la mission est de dresser une liste d'objets susceptibles d'être certifiés et modélisés, et ensuite de déterminer pour ces objets les référentiels de certification adéquats. L'équipe REF m'a assigné sur le projet SPINOV (Software Process Improvement and iNnOVation). Le projet vise à approfondir les compétences du Centre Henri Tudor en évaluation et amélioration des processus d'ingénierie des systèmes d'information et à les développer plus particulièrement en ce qui concerne l'ingénierie des exigences.

L'**objectif** de mon **stage** était de proposer une démarche de tests logiciels adaptée au CITI s'appuyant si possible sur UML (Unified Modeling Language). Mon travail a été décomposé en trois phases : une phase de veille sur les tests logiciels, une phase d'étude des moyens et une phase de proposition d'une démarche de tests. La première phase avait pour objectif d'approfondir mes connaissances sur le thème des tests logiciels et d'identifier les différents types de tests ainsi que les tâches incluses dans la phase de tests dans les divers cycle de vie du logiciel. Durant cette phase, j'ai dû effectuer des interviews pour faire une analyse de l'existant du CITI concernant les pratiques des tests logiciels et pratiquer une évaluation SPICE sur le processus de tests. L'objectif de la seconde phase était d'identifier les moyens mis en oeuvre des tests logiciels concernant les méthodes, les outils ... Et dans un plan plus secondaire, les moyens offerts par UML. La dernière phase avait pour objectif de proposer une démarche de tests logiciels adaptée au CITI s'appuyant sur UML. Cette démarche devait couvrir l'ensemble des tests, définir les exécutants et les responsabilités de la réalisation des tests, s'assurer que les moyens mis en oeuvre sont adaptés à la catégorie de personnes réalisant les tests, et enfin de choisir les moyens les plus appropriés à chaque type de tests.

La **méthode** suivie pour mon **mémoire** est décomposée en trois étapes. La première étape est une recherche sur toute la théorie concernant le champ d'études des tests logiciels et, de manière plus brève, UML. Durant cette étape, j'ai regroupé tous ce qui concerne les tests durant le développement logiciel. Cela va de l'organisation des tests, comme le cycle de vie, les supports disponibles, comme les documents, les activités les plus courantes, les tâches de chaque responsable, les méthodes disponibles ... Suite à cette étape, qui m'a appris beaucoup sur la théorie et la pratique des tests, je me suis demandé comment mettre en oeuvre un processus de tests avec le support d'UML indépendamment du type de projet. Pour terminer, la dernière étape a été de mettre sur pied une démarche de tests logiciels sur un projet en cours de développement.

Suite à cette méthode, mon **mémoire** comporte trois **parties**. La première fait le point sur l'état de l'art concernant les tests logiciels/systèmes et une courte synthèse sur les diagrammes disponibles dans UML. Le premier chapitre s'attarde sur les différents types de tests logiciels et systèmes rencontrés dans la littérature. Le deuxième chapitre étudie la possibilité d'organiser le processus de tests. Le troisième chapitre est une synthèse rapide d'UML et, enfin, le chapitre quatre, fait le point sur le processus de test dans quelques méthodologies de développement et modèles de qualité bien connus. Suite à l'état de l'art, la deuxième partie constitue une modélisation d'un processus de tests pour les projets du CITI. Le cinquième chapitre est une mise en oeuvre du déroulement des grandes classes de tests définies dans la première partie et une proposition d'utilisation de certains diagrammes d'UML pendant le déroulement des tests. Le sixième chapitre représente une synthèse des activités et des supports que l'on peut mettre en oeuvre dans les projets ainsi qu'une modélisation du processus de tests avec UML. La troisième et dernière partie est la démarche de tests réalisée pour un projet en cours de développement durant mon stage au CITI.

Première partie

Etat de l'art

Chapitre 1

Les différents types de tests logiciels

Un système est un « *ensemble de moyens matériels et logiciels mis en oeuvre pour des applications données* » [Lar00].

Un logiciel est un « *programme ou un ensemble de programmes conçus pour le traitement informatique de données* » [Lar00].

Il est important pour la satisfaction de l'utilisateur/client que le développement du logiciel soit de qualité et que le système fasse ce qu'il doit faire. La complexité du développement d'un logiciel augmente proportionnellement avec la taille du projet, les nouvelles technologies mises en oeuvre (l'innovation) et le nombre de personnes impliquées dans sa réalisation. Plusieurs facteurs peuvent entrainer l'apparition d'anomalies, comme l'aspect critique du projet, le nombre de personnes impliquées dans celui-ci, la grandeur du projet qui entraine une plus grande division de celui-ci. C'est pour cela qu'il est très difficile, voire impossible, de ne pas rencontrer d'anomalies dans les différents stades du développement du projet. On peut trouver des anomalies aussi bien dans le code, dans la documentation ainsi qu'à la base, dans la définition des exigences. Il est donc important de planifier des tests variés tout au long du cycle de vie du projet. Les différents tests logiciels vont aider à détecter les anomalies et permettre de les corriger à temps pour qu'il n'y ait pas d'impact sur le fonctionnement du système.

Ce chapitre fait le point sur les types de tests, en les classifiant, effectués pour rechercher les anomalies que l'on peut rencontrer durant le développement du logiciel.

1.1 Classification des anomalies

Une anomalie est définie comme « *un service obtenu par l'exécution du logiciel différent de celui attendu, c'est-à-dire défini par la documentation* » par l'ISO [ISO02].

Un défaut est défini comme « *un élément interne du programme pouvant conduire à l'occurence d'une anomalie lors de l'exécution de ce programme* » par l'ISO [ISO02].

Il est intéressant de classifier et de chercher les anomalies les plus rencontrées lors du développement du système. Les anomalies déjà rencontrées lors d'un projet peuvent aider les futurs développeurs à ne plus répéter les mêmes erreurs ou les aider à comprendre d'où vient les éventuels problèmes. Cette classification des anomalies est mon oeuvre basée sur les anomalies relevées par Pfleeger [Pfl01] et complétées par mes interviews durant mon stage.

1.1.1 Les anomalies algorithmiques

L'anomalie algorithmique survient quand un composant algorithmique ou logique ne produit pas la sortie attendue pour une entrée car quelque chose est faux dans les étapes du processus. On peut citer comme exemple les branchements trop tard ou trop tôt, ou encore une comparaison de variables de types différents.

1.1.2 Les anomalies de syntaxe

L'anomalie de syntaxe arrive lorsqu'on n'utilise pas la bonne syntaxe du langage. Parfois, une erreur de syntaxe qui semble bénigne, peut engendrer un résultat catastrophique. Par exemple, quand on oublie un point virgule dans le programme. Mais généralement, les compilateurs trouvent la majorité des erreurs de ce type.

1.1.3 Les anomalies de précision ou de calcul

L'anomalie de calcul apparaît lorsqu'une formule d'implémentation est fausse ou ne calcule pas le résultat au degré de précision attendu. Le fait de combiner un entier et un réel dans une expression produit un résultat inattendu.

1.1.4 Les anomalies d'intégration

Ce sont des anomalies qui apparaissent lors de l'intégration des composants entre eux.

Dans cette catégorie, on peut encore citer comme sous-catégorie :

a) Anomalie de synchronisation ou de coordination

L'anomalie de coordination ou de séquençage peut survenir quand le code de coordination de ces événements est inadéquat. Lors du développement d'un système, une considération critique est la coordination de quelques exécutions de processus simultanément ou dans une séquence définie avec attention.

1.1.5 Les anomalies fonctionnelles

On parle d'une anomalie fonctionnelle lorsque le système ne produit pas la fonction prescrite par les exigences ou le cahier des charges.

Dans cette catégorie, on peut encore citer comme sous-catégorie :

a) Anomalie dans la documentation

L'anomalie dans la documentation survient lorsque la documentation ne correspond pas à ce que le programme fait réellement. Parfois, la documentation est directement dérivée de l'architecture programme et produit une bonne description sur ce que le programme fait, mais l'implémentation de cette fonction est fausse.

1.1.6 Les anomalies de performance

L'anomalie de performance survient lorsque le système n'a pas la performance en vitesse prescrite ou en ressources par les exigences fonctionnelles et/ou non fonctionnelles.

Dans cette catégorie, on peut encore citer comme sous-catégories :

a) Anomalie de surcharge

L'anomalie de surcharge survient lorsque des limites de longueur de queue, de taille du tampon, par exemple, ont été prévues dans l'architecture et transposées dans les programmes mais que l'utilisation dépasse la limite.

b) Anomalie de capacité

L'anomalie de capacité survient lorsque la performance du système devient inacceptable quand on dépasse les limites spécifiées.

c) Anomalie de reprise

L'anomalie de reprise peut arriver lorsqu'une défaillance est rencontrée et que le système n'a rien de prévu pour surmonter l'obstacle. Par exemple, lors d'une panne de courant durant un traitement, le système doit prévoir une récupération d'une manière acceptable, en restaurant tous les fichiers dans leur état initial avant la coupure.

1.1.7 Les anomalies survenant lors de l'installation du système

Ce genre d'anomalie survient lorsque le système vient d'être installé sur le site client.

Dans cette catégorie, on peut encore citer comme sous-catégories :

a) Anomalie matérielle et logicielle

L'anomalie matérielle ou logicielle peut arriver lorsque les attentes matérielles ou logicielles ne sont pas en accord avec les conditions opérationnelles et procédurales. Par exemple, lorsqu'un nouveau système est mal interfacé avec l'ancien.

b) Anomalie de standard et de procédure

Les anomalies de standard ou de procédure n'affectent pas toujours le déroulement des programmes, mais aboutissent à un environnement dans lequel des erreurs seront créées lorsque le système sera testé et modifié. Le code pourra être revu pour confirmer que les standards et procédures ont été suivis.

1.1.8 Les anomalies de spécification

L'anomalie de spécification peut arriver si une spécification ne traduit pas l'exigence qu'elle devrait décrire. Il faut être bien sûr que les spécifications soient justes car sinon on produit, par exemple, du code juste mais la procédure ne fait pas ce qu'elle devrait faire.

1.1.9 Exemple de classification des anomalies

Chillarege (1992) d'IBM [Pfl01], par exemple, a développé une approche pour pister les anomalies appelée « classification orthogonale des défauts » (voir tableau 1.1- Classification orthogonale des défauts), où les anomalies sont placées en catégories sous forme d'un tableau qui décrit le sens du type d'anomalie. Grâce à ce tableau les développeurs peuvent identifier l'anomalie mais aussi la forme de l'anomalie. Si le développeur est en face d'une anomalie qui affecte l'interface utilisateur, il peut en déduire qu'il est en présence d'une anomalie de type fonction.

Type d'anomalie	Sens
Fonction	Anomalie qui affecte la capacité, l'interface utilisateur, l'interface produit, l'interface avec l'architecture matérielle, ou la structure globale des données
Interface	Anomalie dans l'interaction avec les autres composants ou pilotes via appel, macros, bloc de contrôles, ou listes de paramètres
Contrôle	Anomalie dans la logique du programme
Assignation	Anomalie dans la structure de données ou dans l'initialisation dans le bloc de code
Séquençage/sérialisation	Anomalie qui implique le séquençage de partage et des ressources en temps réel
Version/fusion	Anomalie qui survient quand il y a des problèmes dans les répertoires, le contrôle de version
Documentation	Anomalie qui affecte les publications et la maintenance des notes
Algorithme	Anomalie impliquant l'efficience et l'exactitude des algorithmes ou des structures de données mais pas l'architecture

TAB. 1.1 – Classification orthogonale des défauts

1.2 Les tests logiciels

Plusieurs définitions des tests existent. En voici quelques unes :

« *Un test est la technique de contrôle consistant à s'assurer, au moyen de l'exécution d'un programme, que son comportement soit conforme à des données préétablies* » par l'ISO [ISO02].

« *Un test est une recherche d'anomalie dans le comportement du logiciel* » par Di Gallo [Gal04].

« *Un test est l'activité dont les objectifs sont d'évaluer un attribut ou une capacité d'un système et de déterminer s'il produit ses résultats attendus* » par Hetzel [Pet04].

« *Un test est l'activité qui consiste à exécuter un programme ou un système dans le but d'y découvrir des erreurs* » par Myer [Pet04].

« *Un test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il réponde à ses spécifications ou identifier les différences entre les résultats obtenus* » par IEEE [Kon04].

On peut définir un composant comme « *représentant un bout de logiciel qui se conforme à un ensemble de services* » [Enc04].

Bertolino définit le test comme « *une technique pour évaluer la qualité du produit et, indirectement, l'améliorer, en identifiant les défauts et les problèmes* » [Ber04].

L'objectif des tests est de réduire les risques d'apparition d'erreurs dans le système en utilisant des moyens manuels et informatiques. Il y a beaucoup de tests différents qui sont effectués tout au long du développement du logiciel, et qui ont pour but de détecter une anomalie bien précise. Ils sont regroupés en de grands groupes de tests rencontrés dans la littérature, par différents auteurs.

1.2.1 Une vue des tests par aspects testés (Pfleeger)

Selon Pfleeger [Pfl01], (voir la figure 1.1 - Vue des tests par aspects) on doit procéder à des tests unitaires sur tous les composants. Lorsque tous les composants ont passés les tests unitaires, on procède aux tests d'intégration en s'aidant des spécifications d'architecture. Ensuite, on pratique des tests fonctionnels pour voir si toutes les fonctions demandées dans les exigences fonctionnelles existent. Après cela, des tests de performance vérifient que le logiciel respecte les exigences non-fonctionnelles. Les utilisateurs procèdent ensuite à des tests d'acceptation.

Et enfin, il faut effectuer des tests d'installation sur le site dans lequel le logiciel sera mis en route.

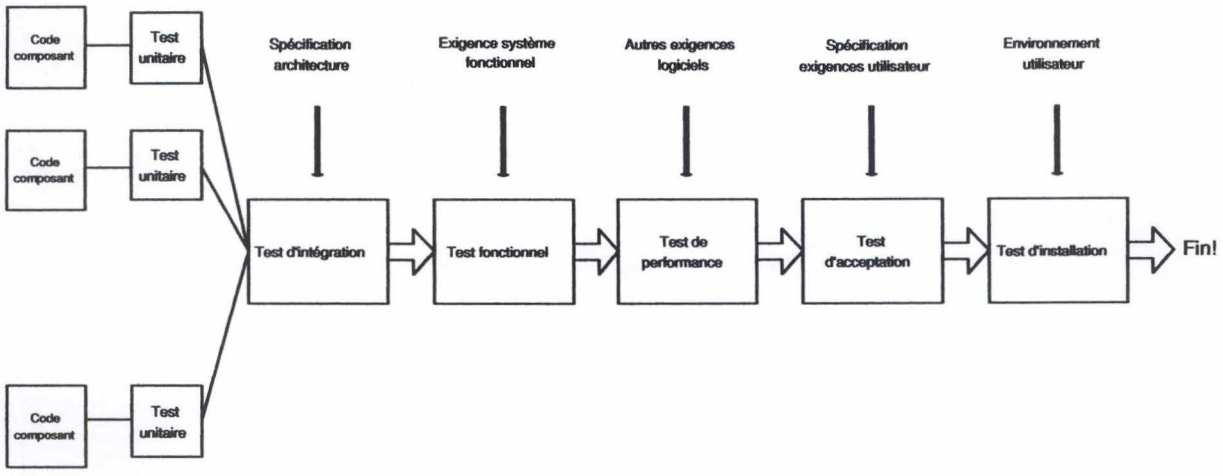


FIG. 1.1 – Vue des tests par aspects

1.2.2 Une vue des tests comme inspection/qualité (Gilb)

Pour Gilb et Graham [GG98], (voir la figure 1.2 - Vue des tests comme inspections) un test « ***vise à améliorer la qualité du logiciel. Le but est de trouver et de fixer les erreurs, les défauts et d'autres problèmes potentiels.*** »

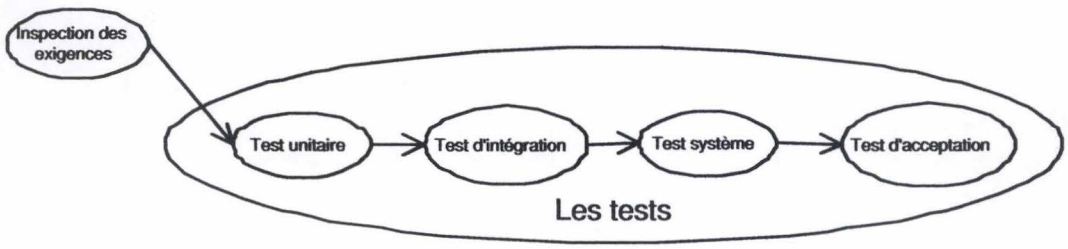


FIG. 1.2 – Vue des tests comme inspections

Les auteurs donnent quelques pratiques professionnelles en ce qui concerne les tests logiciels :

Des pratiques de préparation :

- Le planning et l'architecture des tests doivent être pensés au début du cycle de développement du logiciel ;
- Les tests d'acceptation sont définis par les utilisateurs en même temps que l'utilisateur définit les exigences ;
- Les tests systèmes sont définis durant l'analyse des exigences ;
- Les tests d'intégration sont définis durant l'architecture ;
- Les tests unitaires sont définis durant l'architecture détaillée ;

Des pratiques d'exécution :

- Exécution des tests unitaires ;

- Exécution des tests d'intégration ;
- Exécution des tests systèmes ;
- Exécution des tests d'acceptation ;

Une pratique d'évaluation :

- Evaluation de l'efficacité des tests.

Il faut définir les tests avant le développement du cycle de vie (Hetzel, 1984). Pour les auteurs, l'inspection des exigences permet déjà de trouver des erreurs, des défauts ou des problèmes et, ainsi, de diminuer les coûts liés aux tests.

1.2.3 Une vue des tests par niveaux (Bertolino)

Pour Bertolino [Ber04], les tests sont distingués en trois grands niveaux : les tests unitaires, les tests d'intégration et les tests systèmes (voir la figure 1.3 - Vue des tests par niveaux).

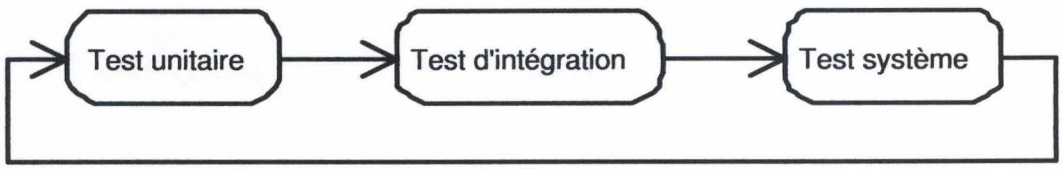


FIG. 1.3 – Vue des tests par niveaux

Il faut d'abord procéder à des tests unitaires, puis à des tests d'intégration et pour terminer à des tests systèmes. C'est-à-dire que l'on vérifie d'abord le fonctionnement composant par composant, puis on vérifie l'interaction entre les différents composants entre eux et enfin, on vérifie le comportement du système tout entier.

1.2.4 Une vue des tests par dimensions (RUP)

Dans le RUP [Rat01], les types de tests sont classés par leur objectif ou leur dimension qualité. On trouve trois dimensions qualités qui sont la fiabilité, la fonctionnalité et la performance.

Dans la dimension fiabilité on trouve les tests suivant :
- Tests de robustesse voir le point 1.3.4 Les tests de performance.

Dans la dimension fonction, on trouve :
- Test de configuration voir le point 1.3.6 Les tests d'installation.
- Test fonctionnel voir le point 1.3.3 Les tests fonctionnels.
- Test d'installation voir le point 1.3.6 Les tests d'installation.
- Test de sécurité voir le point 1.3.4 Les tests de performances.
- Test de volume voir le point 1.3.4 Les tests de performances.

Dans la dimension performance, on trouve les tests suivant :
- Test de performance voir le point 1.3.4 Les tests de performances.
- Test de collision ou de contention voir le point 1.3.4 Les tests de performances.
- Test de chargement voir le point 1.3.4 Les tests de performances.
- Test de séquençage voir le point 1.3.4 Les tests de performances.
- Test de surcharge voir le point 1.3.4 Les tests de performances.

Les tests sont effectués tout au long du développement sur différentes cibles et à des étapes différentes. Les étapes du processus commencent par tester les plus petits composants, tests unitaires, jusqu'aux tests du système complet, les tests système.

1.3 Proposition de classification des types de tests

J'ai finalement opté pour la classification de Pfleeger car elle donne directement l'aspect du logiciel ou du système qui est testé, sans se préoccuper des différentes sortes. Elle est aussi plus complète que les autres et applicable pour tout projet. Ensuite, j'ai regroupé tous les tests rencontrés dans la littérature, limité à ce que j'ai analysé durant mon stage, dont la plupart dans [Dan04], [Pfl01] et [Rat01], sous cette classification. Evidemment, ce choix est personnel. Il faut signaler qu'un test peut avoir différents aspects en fonction du contexte. Par exemple, le test de sécurité que l'on peut voir comme un aspect fonctionnel, si la sécurité implique des fonctionnalités explicites, ou de performance.

Les points suivants représentent la classification que je propose.

1.3.1 Les tests unitaires

Les tests unitaires consistent à tester individuellement les composants du logiciel. On pourra ainsi valider la qualité du code et les performances d'un composant. On vérifie pour chaque composant du logiciel que les résultats sont conformes au dossier de conception détaillée, que la plage de validité des entrées/sorties a été explorée et que tous les chemins sont parcourus.

1.3.2 Les tests d'intégration

Les tests d'intégration sont exécutés pour valider l'intégration des différents composants entre eux et dans leur environnement d'exploitation définitif. Ils permettront de mettre en évidence des problèmes d'interfaces entre les différents composants, par exemple de synchronisation ou de séquençage entre les différents composants. Le but est de vérifier que chaque agrégat (ensemble de composants) soit conforme aux exigences attendues et que tous les appels de composants soient parcourus.

Dans cette catégorie, on peut encore citer comme sous-catégorie :

a) Les tests système

Les tests système permettent de tester un système d'éléments matériels et logiciels intégrés afin de prouver que le système réponde aux exigences spécifiées dans le cahier des charges ou dans les spécifications.

1.3.3 Les tests fonctionnels

Les tests fonctionnels ont pour but de vérifier la conformité de l'application développée avec le cahier de charge initial. Ils sont donc basés sur les spécifications fonctionnelles et techniques. Le but est de vérifier que toutes les fonctionnalités demandées soient implémentées et font ce qu'elles doivent faire.

Dans cette catégorie, on peut encore citer comme sous-catégorie :

a) Les tests d'interface homme machine

Les tests IHM ont pour but de vérifier que la charte graphique et les règles ergonomiques, aient été respectées tout au long du développement du logiciel.

1.3.4 Les tests de performances

Le but principal des tests de performances est de valider la capacité qu'a le système à supporter des charges d'accès importantes. Le deuxième objectif de ces tests est de valider le comportement du logiciel, toujours dans des conditions extrêmes. Ces tests doivent permettrent de définir un environnement système optimal pour que le logiciel fonctionne correctement.

Dans cette catégorie, on peut encore citer comme sous-catégories :

a) Les tests de non-régression

Les tests de non-régression vérifient en rejouant les tests déjà réalisés que le logiciel n'a pas été dégradé après une modification, c'est-à-dire l'ajout ou l'adaptation d'une fonction ou la correction d'une anomalie.

b) Les tests de surcharge

Les tests de surcharge évaluent le système à ses limites pendant une petite période. Le système est testé avec des conditions anormales pour analyser le comportement du système. Par exemple, pour détecter une insuffisance de mémoire ou encore, un service ou un matériel indisponible.

c) Les tests de volume

Les tests de volume font un maniement d'un large nombre de données dans les bases de données du système et voient comment se comporte le système par exemple, en implémentant une requête qui demande l'entièreté des données de la base de données.

d) Les tests de sécurité

Les tests de sécurité assurent que les exigences de sécurité soient au rendez-vous. Par exemple, pour s'assurer de l'intégrité ou la confidentialité des données.

e) Les tests de séquençage

Les tests de séquençage évaluent les exigences transactionnelles avec le temps de réponse pour un utilisateur et le temps pour faire la fonction. Ce test est souvent couplé avec le test de surcharge.

f) Les tests de reprise

Les tests de reprise testent les moyens mis en oeuvre pour la reprise lorsqu'une panne ou une perte de données intervient. Après une panne électrique, par exemple.

g) Les tests de documentation

Les tests de documentation assurent que les documents des exigences existent. C'est-à-dire que toute la documentation a bien été écrite. Par exemple, que les exigences ou encore la documentation des utilisateurs existent.

h) Les tests des facteurs humains

Les tests des facteurs humains étudient les exigences des transactions avec l'interface utilisateur du système.

i) Les tests de robustesse

Les tests de robustesse sont des tests à exécuter pour vérifier que le logiciel continue à fonctionner correctement en dépit de l'introduction d'entrées anormales.

j) Les tests de parallélisme

Le test de parallélisme est mis en oeuvre lorsque le nouveau système opère en parallèle avec un ancien système. L'utilisateur utilise encore l'ancien système, et utilise petit à petit le nouveau. Cela permet à l'utilisateur de bien faire la comparaison entre l'ancien et le nouveau ou permet une transition graduelle vers le nouveau système.

k) Les tests de collision ou de contention

Le test de collision assure que la cible du test puisse accepter des accès multiples de différentes demandes sur la même ressource. Par exemple, lors d'enregistrement de données.

l) Les tests de chargement

Le test de chargement vérifie et évalue l'acceptabilité des limites opérationnelles d'un système sous différents travaux quand le système testé est constant.

1.3.5 Les tests d'acceptation

Les tests d'acceptation testent le système dans les conditions définies par les utilisateurs.

Dans cette catégorie, on peut encore citer comme sous-catégorie :

a) Les tests de recette

Les tests de recette doivent confirmer que le système répond d'une manière attendue aux requêtes qui lui sont envoyées.

1.3.6 Les tests d'installation

Une fois l'application validée, il est nécessaire de contrôler les aspects liés à l'installation. Les procédures d'installation doivent être testées intégralement car elles garantissent la fiabilité de l'application dans la phase de démarrage, par exemple, en testant le comportement du système face à une insuffisance de disque.

Dans cette catégorie, on peut encore citer comme sous-catégories :

a) Les tests pilotes

Ce test a pour but d'installer le système sur une base expérimentale. L'utilisateur exerce le système comme s'il était installé en permanence. Le test pilote est moins formel que le test de performance.

b) Les tests de configuration

Les tests de configuration analysent les différentes configurations logicielles et matérielles spécifiées dans les exigences. Le test évalue toutes les possibilités pour être sur que chacune satisfasse les exigences.

c) Les tests de compatibilité

Les tests de compatibilité sont nécessaires lorsque plusieurs systèmes sont interfacés entre eux. Ce test vérifie si les différents matériels et logiciels interfacés sont compatibles.

d) Les tests environnementaux

Les tests environnementaux regardent les capacités du système à évoluer dans le site d'installation.

1.4 Conclusion

Le développement d'un logiciel sans anomalie n'existe pas. Toutes sortes d'anomalies peuvent se produire, d'un simple oubli d'un point virgule dans une ligne de code, à une importante anomalie de fonctionnement. Chaque anomalie non détectée et non corrigée peut produire une ou des erreurs sur le fonctionnement du système. Afin de diminuer le risque d'erreur dans le logiciel, on doit pratiquer divers tests sur le logiciel durant tout son développement. Plus les tests sont diversifiés et spécifiques, moindre est le risque de produire des erreurs dans le logiciel. La production d'un historique des anomalies rencontrées durant le projet est utile pour ne plus répéter les erreurs d'autrefois et aider les développeurs à trouver plus facilement la source de l'anomalie. Les tests ont pour objectif la prévention contre les erreurs, plutôt que leur correction.

Chapitre 2

L'organisation des tests logiciels

Organiser les différents tests pour produire des logiciels de qualité est un élément important pour le processus de tests. Une bonne organisation peut permettre à l'équipe de test de savoir ce qu'elle doit faire, pourquoi elle le fait et, enfin ce qu'elle fait. Dans l'organisation du processus, on trouve souvent un cycle de vie, une stratégie, de la documentation et enfin, des outils. Il ne faut pas oublier de bien définir les rôles et les responsabilités de chaque personne en composant une ou plusieurs équipes de tests.

Ce chapitre s'attarde sur l'organisation du processus de tests c'est-à-dire le cycle de vie, la stratégie, la documentation, l'équipe à mettre en place et enfin les outils de tests automatiques.

2.1 Le cycle de vie des tests

Un cycle de vie est défini comme « ***un cadre de travail comprenant les processus, les activités, et les tâches impliquées dans le développement, le fonctionnement et la maintenance d'un produit logiciel, couvrant la vie du système depuis la définition de ses exigences jusqu'à la fin de son utilisation*** » par l'ISO 12207 [HA04].

On peut reprendre la même définition pour le cycle de vie des tests logiciels, en indiquant qu'ici, c'est la vie du processus de tests logiciel. Ce cycle de vie est un outil de gestion de projet qui permet d'identifier les différentes phases des tests, les activités et les délivrables de chaque phase identifiée, les liens entre les phases et les activités, les rôles et les responsabilités des personnes impliquées dans les tests.

2.1.1 Exemple d'un cycle de vie des tests logiciels : RUP

Le cycle de vie des tests dans le RUP (voir figure 2.1 - Cycle de vie des tests dans RUP) est une approche itérative. Les tests se passent tout au long du développement en effectuant des raffinements et des additions aux jeux de tests. Dans ce type d'approche les spécifications et les jeux de tests ne sont pas fixes. Lors de chaque itération, on effectue des tests de non-régression pour détecter les anomalies possibles dûes au changement ou à l'addition de composants.

Le cycle de vie se présente comme suit : la planification des tests, la conception des tests, l'implémentation des tests, l'exécution des tests en les intégrant et l'évaluation des tests. Comme l'approche est itérative, on retrouve ce schéma lors de chaque itération.

Description des étapes :

- La planification des tests comprend les exigences et la stratégie de test.
- La conception des tests permet d'identifier, de décrire et de programmer le modèle, la procédure et les jeux de test.

- L'implémentation des tests sert à enregistrer, générer et programmer la procédure de test pour obtenir le script des tests.
- L'exécution des tests en les intégrant assure que la collaboration entre composants se comporte bien. A cette étape, on exécute les tests de non-régression.
- L'évaluation des tests génère et délivre un résumé de l'évaluation des tests qui comporte le résultat et les mesures clefs des tests.

Chaque activité communique avec la suivante soit en passant des documents ou composants utilisés.

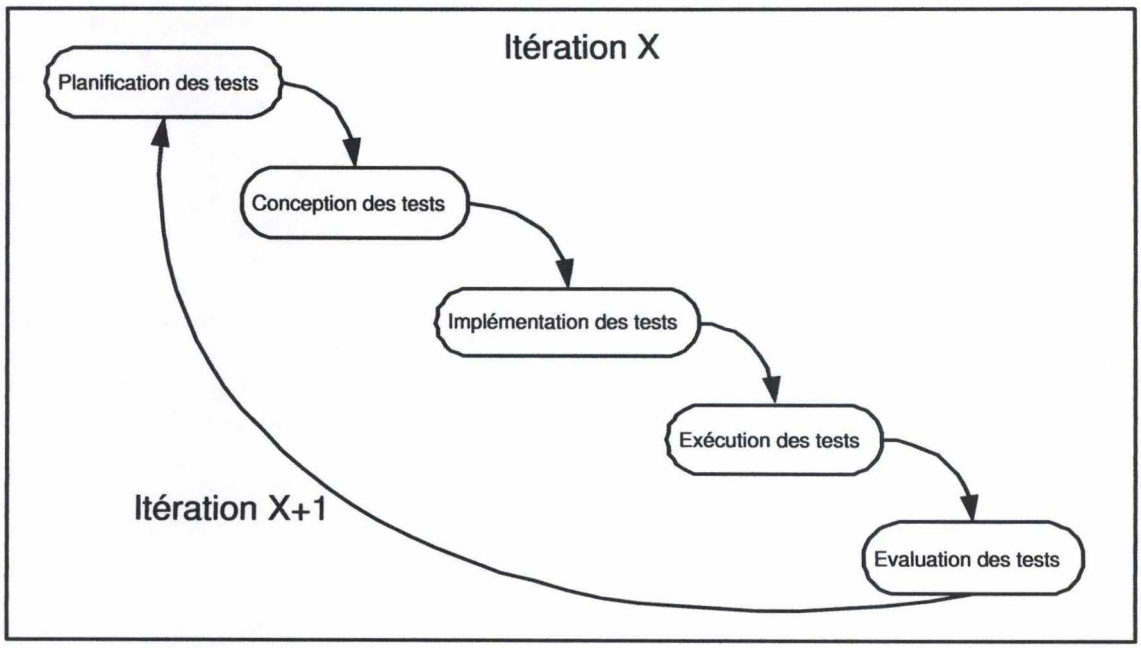


FIG. 2.1 – Cycle de vie des tests dans RUP

2.1.2 Exemple d'un cycle de vie des tests logiciels : le modèle en V

La particularité du cycle de vie en V est que toute description d'un composant est accompagnée de tests spécifiques, ce qui permet de s'assurer qu'il corresponde bien à sa description (voir la figure 2.2 - Cycle de vie en V). Cela permet aussi l'obligation de concevoir les jeux de tests et leurs résultats, ainsi qu'une meilleure préparation de la branche de droite du V [Tol04] et [Anc04].

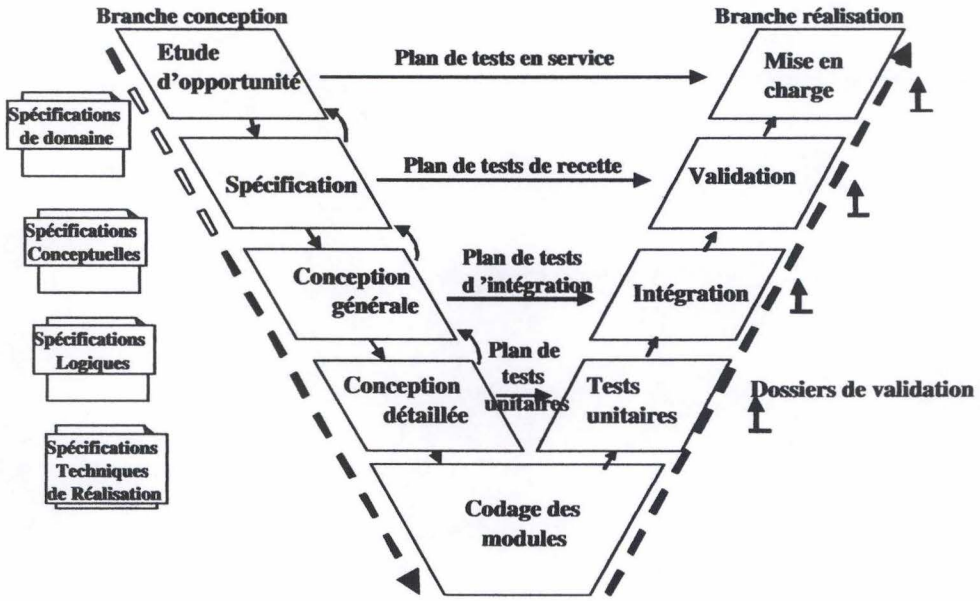


FIG. 2.2 – Cycle de vie en V

Ce cycle de vie en V signale qu'il faut faire le plan de tests pendant la phase d'étude d'opportunité et qu'il sera utilisé pendant la phase de mise en charge du logiciel. Le plan de recette doit être fait pendant la phase de spécification du logiciel et sera utilisé pendant la phase de validation. Et ainsi de suite ...

2.2 La stratégie de tests

Un jeu de tests ou cas de tests peut être défini comme « *un chemin fonctionnel à mettre en oeuvre pour atteindre un objectif de test. Un cas de test se définit par le jeu d'essai à mettre en oeuvre, le scénario de test à exécuter et les résultats attendus* » [Dic04].

La stratégie de tests aide à concevoir et à organiser les tests, d'une manière telle qu'on est sûr de faire les tests appropriés au moment approprié.

Chaque étape des tests doit être planifiée et doit avoir lieu en parallèle avec le développement du logiciel. Selon Pfleeger [Pfl01], il faut planifier les tests comme suit :

- établir les objectifs des tests, quelle sorte de test produire
- concevoir les jeux de tests
- écrire les jeux de tests
- tester les jeux de tests, vérifier s'ils sont corrects, faisables et leur couverture
- exécuter les tests
- évaluer les résultats des tests

Les objectifs des tests sont très importants dans la stratégie. Ils dévoilent quel type de jeux de tests est à générer. La conception des jeux de tests est la clef du succès des tests. Si les tests sont non représentatifs des exigences/besoins, alors ils ne servent à rien.

2.3 La documentation des tests

Pour assurer une bonne gestion des tests, on utilise une documentation des tests. Plusieurs documents peuvent être établis :

- Un plan de test est un document décrivant la spécification, l'environnement ainsi que la planification des tests. L'objectif du plan de test est d'organiser les activités des tests. Il incorpore la stratégie et les échéances des tests.
- Une spécification et une évaluation des tests est un document de description des principes de conception et d'élaboration des jeux de tests.
- Une description des tests est un document qui présente la façon de mettre en oeuvre la spécification et la façon de conduire les tests.
- Une analyse du rapport de tests qui décrit les résultats de chaque test.

Les documents servent aussi à contrôler si toutes les étapes dans les tests ont bien été effectuées. Ils peuvent donc constituer une preuve que l'activité a bien été menée par la personne responsable.

2.4 L'équipe de tests

Shari Lawrence Pfleeger [Pfl01] préconise de former une équipe de tests dès le début du développement du projet. Une équipe qui soit formée de personnes ayant des connaissances variées, comme un ingénieur matériel, un concepteur, un développeur et une personne du domaine en question. En plus de cela, elle préconise que l'équipe soit indépendante de l'équipe du projet pour avoir une vue externe.

Si le projet a un aspect peu critique et qui ne peut pas supporter trop de frais, on préférera former une équipe interne.

Si, par contre, le projet est très critique, on préférera faire appel à une équipe externe/indépendante spécialisée dans les tests (culture d'entreprise) et maîtrisant les connaissances.

2.5 Les outils de tests automatiques

Le processus de tests peut être facilité par l'aide d'outils logiciels de tests automatiques. Parmi les outils de test automatique, on peut trouver trois types d'outils qui sont soit des outils d'analyse de code, soit des outils d'exécution de test, soit des générateurs de jeux de tests [Pfl01].

Il existe deux catégories d'outils d'analyse de code, qui sont l'analyse statique et l'analyse dynamique. L'analyse statique est effectuée lorsque le programme n'est pas encore exécuté. Il exécute quatre types de vérifications qui sont une analyse de code, une vérification de la structure, une analyse des données et une vérification des séquences. L'analyse dynamique, par contre, est sollicitée lorsque le programme tourne. On trouve dans cette catégorie d'outil les programmes de surveillance qui permettent à l'équipe de tests de capturer les états des événements durant l'exécution du programme, en conservant une copie d'écran des conditions et des chemins d'exécution.

Un outil d'exécution de tests est un outil de capture et de réinsertion. Il mémorise les frappes, les entrées et les réponses quand les tests sont en route. Ils comparent les sorties

attendues avec celles que les tests délivrent. Un outil d'exécution peut être aussi un outil qui assiste l'informaticien dans la création de routines et de pilotes automatiques.

Un générateur de jeux de tests génère des classes de possibilités de situation dans les jeux. Il existe plusieurs types de générateur de jeux des tests dont le générateur structurel qui base ses jeux de tests sur la structure du code source. On peut trouver d'autres générateurs basés, par exemple, sur le flux de données, ou encore sur les tests fonctionnels.

Plusieurs raisons existent pour automatiser des tests d'une application logicielle. Tout d'abord, ils permettent de réduire le temps requis pour tester. Ceci est d'autant plus vrai, lorsque le testeur doit rejouer les mêmes jeux de tests plusieurs fois, par exemple, lors des tests de non-régression qui sont répétitifs et réutilisés. Une autre raison liée à la première, est que la productivité du testeur est améliorée. Une autre raison est que la qualité de l'application développée est améliorée car un test automatique est toujours exécuté de la même façon alors qu'un test manuel ré-exécuté pourrait varier. Comme le testeur perd moins de temps avec les tests automatiques, il peut alors produire des tests plus profonds et mieux documentés.

Selon la CRIM [CRI04], automatiser un test permet de réduire la durée des tests à l'exécution jusqu'à 60%. Mais il coûte jusqu'à dix fois plus cher à préparer qu'un test manuel. De plus, plusieurs études démontrent que le nombre de défauts par ligne de code livrée au client décroît de 10% à 30% comparé aux tests manuels.

2.6 Les coûts liés aux tests

Le coût pour réparer une erreur, augmente tout au long du projet (Boehm, 1981) [GG98]. Soit le coût pour trouver une erreur pendant les exigences est égale à un, les chiffres obtenus sur soixante trois projets ont montré que :

Erreur trouvée pendant	Coût
Exigence	1
Architecture	3x-6x
Codage	10x
Test de développement	15x-40x
Test d'acceptance	30x-70x
Opération	40x-1000x

TAB. 2.1 – Coût pour corriger une erreur

Il faut donc tester le plus tôt possible, ce qui se traduit par des coûts moindres en ce qui concerne les tests.

2.7 Conclusion

Une bonne organisation des tests logiciels passe par une bonne description d'un cycle de vie des tests que l'on va effectuer sur le logiciel pendant son développement, une stratégie des tests, des documents et des outils d'aide adaptés. Plus tôt les tests sont pratiqués, moins la détection des erreurs coûte cher.

Le cycle de vie va nous renseigner sur les activités de test à effectuer et leur interconnexion. La stratégie donne la direction que les tests vont prendre selon les objectifs de ceux-ci. La documentation de suivi peut jouer un rôle de preuve ou de traçabilité que les tests ont bien été effectués et dans quelles conditions. Les outils peuvent aider l'équipe de tests dans leur lourde tâche surtout lorsque les tests se répètent.

Il est clair que le cycle de vie doit être décidé avant que les tests ne commencent et doit être suivi tout au long du développement du logiciel. En ce qui concerne l'équipe de test, elle peut être interne ou externe au projet dépendant des frais, du temps, de la culture d'entreprise et enfin, de la qualité recherchée du projet.

La qualité recherchée dépend de l'aspect critique du projet. Dans un projet très critique, on préférera une qualité zéro défaut avec une équipe indépendante/externe et, pour un projet peu critique, une équipe interne pour rechercher le plus de défauts possibles.

Chapitre 3

Unified Modeling Language

Ce chapitre est une petite synthèse sur UML (version 1.4) et ses diagrammes.

3.1 Présentation d'UML

UML (Unified Modeling Language) est un langage permettant de décrire des modèles d'un système basé sur des concepts orienté objets. Ce langage est né d'une fusion des méthodes BOOCHT, OMT (Object Modeling Technique) et OOSE (Object Oriented Software Engineering). En 1997, UML devient une norme OMG (Object Management Group).

Ce langage permet de spécifier, visualiser et comprendre le problème, de capturer, communiquer et utiliser des connaissances pour la résolution de problèmes, de spécifier, visualiser et construire la solution, et enfin de documenter la solution.

Modéliser avec UML, c'est créer plusieurs vues du système pour permettre aux différents acteurs du développement de voir le système de différentes manières. UML n'est donc pas une méthode mais une notation, un langage. UML, permet de modéliser les exigences fonctionnelles par les scénarii dans les cas d'utilisation, de concevoir l'architecture par les diagrammes d'activités, d'états, de classe, d'objets, de séquences et de collaboration, et enfin de coder par les diagrammes de composants et de déploiement [FL00a] et [FL00b].

3.2 Brève présentation des diagrammes

3.2.1 Les diagrammes de cas d'utilisation

Le cas d'utilisation peut être utilisé tout au long du développement du logiciel. Il donne dans un premier temps les exigences, d'une manière informelle, que le système devra satisfaire. Grâce aux opérations de composition et de spécialisation, auxquelles des variantes peuvent être développées (scénario). Le cas d'utilisation peut aussi servir dans la formalisation des comportements et de la synchronisation des actions via les pré et post-conditions. Il constitue le point d'entrée pour définir un périmètre fonctionnel.

Avec les scénarii, les pré- et post-conditions, le cas d'utilisation constitue un squelette du plan de test.

3.2.2 Les diagrammes de séquences

Le diagramme de séquence fournit l'axe central de toute la modélisation des interactions. Ce diagramme représente les participants et leurs responsabilités, la séquence des messages, activités et synchronisations, la structuration des traitements, et les contraintes d'exécution. Il permet de reprendre les éléments du diagramme de collaboration avec un formalisme plus rigoureux, en particulier pour la représentation du temps.

3.2.3 Les diagrammes d'activités

Ce diagramme ne modélise pas les interactions, mais permet de modéliser les activités simultanées. Il permet de modéliser directement les cas d'utilisation avec leurs variantes.

3.2.4 Les diagrammes d'états/transitions

Ce diagramme est utilisé pour formaliser des comportements dirigés par les états ainsi que pour modéliser le déroulement des activités dans leur contexte organisationnel et technique.

3.2.5 Les diagrammes de collaboration

Le diagramme de collaboration décrit les échanges de messages entre classes et définit les associations. Il est équivalent au diagramme de séquence. Les différences entre les deux diagrammes sont que le diagramme de collaboration n'illustre pas l'ordre des événements mais qu'il représente les interconnexions entre objets. Il permet de modéliser les interactions entre objets.

3.2.6 Les diagrammes d'objets

Ce diagramme est utilisé pour décrire l'état du système à un instant donné. On l'emploie souvent pour documenter les pré- et post-conditions.

3.2.7 Les diagrammes de classe

Le diagramme de classe donne une vue statique du système, à l'aide de classes, de paquetages et de leurs relations. Il est utilisé pour la description des types dans un modèle d'analyse et pour décrire l'implémentation des types dans un modèle de conception.

3.2.8 Les diagrammes de composants

Ce diagramme décrit l'organisation des composants et les dépendances qui les lient. Il détermine aussi l'architecture applicative du système en termes de composants et d'interfaces. Il permet de configurer l'environnement d'exploitation.

3.2.9 Les diagrammes de déploiement

Le diagramme de déploiement décrit les ressources de calcul, leur configuration et le lien entre exécutables et les ressources de calcul. Il sert à combiner les composants et les moyens techniques nécessaires à leur exploitation.

3.3 Les stéréotypes

Les cinq stéréotypes (types d'entités, interfaces, contrôleurs, acteurs et descriptifs) permettent, dans un premier temps, d'adapter UML au contexte applicatif et méthodologique des différentes entreprises et, dans un deuxième temps, de formaliser et de systématiser la transition entre phase d'analyse et de conception. Ils offrent une extension du langage [FL00a].

3.4 Conclusion

UML est un langage qui permet de modéliser les différentes parties du projet et du système via ses diagrammes et ses stéréotypes. Les différentes modélisations aident l'équipe projet à mieux visualiser, spécifier et construire le projet que ce soit dans la phase d'analyse des exigences ou de conception.

Chapitre 4

Le processus de test dans quelques processus logiciel

Un processus est défini comme « *un ensemble d'activités en interaction, qui transforme des entrées en sorties* » [Ber04].

Un processus est défini, par N. Habra et A. Renault, comme « *un ensemble structuré de pratiques nécessaires à la réalisation d'un objectif commun clairement défini* » [HR04].

Une pratique est définie, par N. Habra et A. Renault, comme « *une activité d'ingénierie qui contribue à la réalisation de l'objectif d'un processus par la création d'un délivrable ou l'amélioration de la capacité du processus* » [HR04].

Pour Bertolino, « *le processus de tests définit des activités de test et permet de conduire les équipes de test, du plan de test à l'évaluation des résultats de test, pour assurer que les objectif de test rencontrent des coûts raisonnables* » [Ber04].

Pour Habra et Renault, « *le processus de tests a pour but de vérifier l'adéquation du produit logiciel par rapport aux exigences, et de détecter les erreurs* » [HR04].

Le but d'un processus de tests est bien de guider une équipe de test dans les tests à produire, en donnant des indications sur les activités ou les pratiques à effectuer, et les documents à fournir ou à remplir en vue de vérifier l'adéquation du logiciel par rapport aux exigences et de détecter les erreurs ou les anomalies.

Ce chapitre s'attarde sur le processus de tests que l'on trouve dans quelques méthodologies de développement, qui incluent une modélisation du développement, les plus connues comme RAD, XP, Scrum, FDD, DSDM et RUP, et sur quelques modèles de qualité bien connus, qui incluent une modélisation du développement, comme CMM, SPICE et CMMI.

4.1 Rapid Application Development

4.1.1 Présentation générale

La méthode Rapid Application Development (RAD) et le Processus Qualité RAD2, impliquent trois intervenants principaux qui sont la maîtrise d'oeuvre, la maîtrise d'ouvrage ainsi que le groupe d'animation et de rapport [Vic04].

Une des grandes phases de la maîtrise d'oeuvre est tout d'abord l'initialisation. Cette phase définit l'organisation, le périmètre et le plan de communication. Ensuite vient le cadrage. Ici, on définit un espace d'objectifs, de solutions et de moyens. La spécification des exigences est du ressort des utilisateurs. La troisième phase est la conception. Durant cette phase, on modélise la solution et on valide sa cohérence systémique. Les utilisateurs sont également impliqués

dans cette étape. Pendant la construction, la phase suivante, on réalise en prototypage actif c'est-à-dire une validation permanente. L'équipe RAD doit construire l'application, composant par composant. L'utilisateur participe toujours activement aux spécifications détaillées et à la validation des prototypes. La dernière phase est la finalisation. Ici, on réalise un contrôle final de qualité en site pilote. Des recettes partielles ayant été obtenues à l'étape précédente, il s'agit dans cette phase d'officialiser une livraison globale et de transférer le système en exploitation et maintenance.

Le projet est piloté selon un suivi rigoureux des contraintes, des risques et de la qualité technique. La maîtrise d'ouvrage doit assurer l'expression des exigences et sa validation permanente, la préparation au changement organisationnel, la recette fonctionnelle et technique et le démarrage du projet. Il est aussi piloté selon un suivi rigoureux de la qualité fonctionnelle par la mise en oeuvre de rapports de focus [1] et des suivis des divergences. Le groupe d'animation et de rapport prend en charge les communications et la formalisation des informations.

4.1.2 Processus de tests

A chaque prototypage, on pratique les tests unitaires sur les composants. Ensuite, on intègre tous les composants en effectuant les tests d'intégration. En même temps que les tests d'intégration, on procède à des tests d'interface homme-machine (IHM) suivant une charte graphique. Lorsque tous les tests sont finis et produisent les résultats attendus, on effectue alors des tests fonctionnels.

Si un ou plusieurs composants changent durant le prototypage, il faut procéder alors à des tests de non-régression et refaire les tests d'intégration complets.

4.2 Extreme Programming

4.2.1 Présentation générale

L' Extreme Programming (XP) est une approche réfléchie et disciplinée du développement logiciel qui met l'accent sur la satisfaction du client. Elle permet de livrer le logiciel en temps voulu. Cette méthode a été mise en oeuvre pour la première fois en 1996, par Kent Beck, sur le projet d'un nouveau système de paie pour les salariés de l'entreprise Chrysler [HA04].

L'XP est préconisée [Mar03] dans un milieu où les besoins changent. Pour faire face aux risques liés aux projets et augmenter leurs chances de succès. Cette méthodologie est adaptée aux petits groupes de projet comprenant de 2 à 10 personnes ou un cycle de développement court. Les petites équipes sont plus efficaces que les grosses lorsque le risque est grand et les besoins dynamiques. Pour essayer de rester dans le planning, XP simplifie les règles, et garde celles qui contribuent à la qualité tout en laissant de côté celles qui ralentissent le projet.

Cette méthode allégée comprend six phases [eP04] qui sont l'exploration, la planification, l'itération de la version, la production, la maintenance et, enfin, la finalisation (voir la figure 4.1 - La méthode XP).

[1] réunion plénière de présentation débouchant sur une validation globale [Vic04]

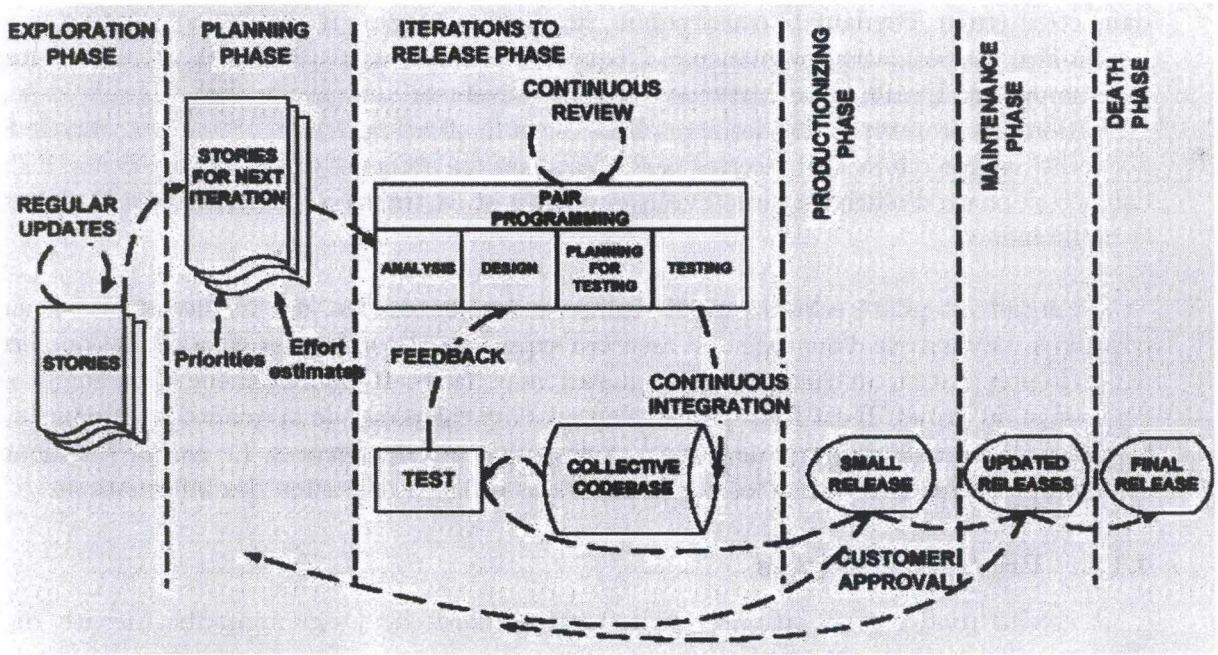


FIG. 4.1 – La méthode XP

4.2.2 Processus de tests

Dans cette méthode, la phase de test prévoit des tests unitaires, des tests fonctionnels et un traitement des anomalies. Les tests sont définis avant l'écriture du code.

L'accomplissement des tests unitaires nécessite de tester toutes les classes du système. Les tests unitaires sont livrés dans le dépôt (sous forme de répertoire réservé aux tests) des codes sources accompagnés du code qu'ils testent. Si un test unitaire n'est pas trouvé alors il doit être conçu. Ainsi, l'ensemble des tests est disponible au moment voulu par toute l'équipe. Ce genre de test favorise la possession collective du code et la refonte du code. Lorsque l'on ajoute une nouvelle fonctionnalité, cela nécessite le plus souvent de changer les tests unitaires.

Le test fonctionnel, qui est le plus souvent automatisé (recommandé par XP), est un système de tests boîte noire qui valide ou non les changements intervenus précédemment. Ces tests sont générés à partir des scénarii d'utilisateur. La vérification de leur correction est de la responsabilité des utilisateurs. Le résultat est diffusé à toute l'équipe qui planifie le temps nécessaire à la correction des anomalies, pour chaque itération. Dans la phase de production, l'équipe vérifie la performance du système avec des tests supplémentaires.

Lorsqu'une anomalie est trouvée, il faut créer un test pour éviter la prolifération de ce type d'anomalie. On entre alors dans le traitement des anomalies. Avant le « déboguage », il est préconisé de créer un test fonctionnel pour aider l'utilisateur à déterminer le problème et ensuite le communiquer aux développeurs. Si les tests fonctionnels échouent, on procède alors à des tests unitaires.

On trouve deux rôles spécifiques aux tests : les testeurs et les programmeurs [ASRW02]. Les testeurs aident les utilisateurs à écrire les tests fonctionnels. Ils appliquent les tests régulièrement, diffusent les résultats des tests et maintiennent les outils de tests. Les programmeurs écrivent les tests.

On trouve aussi deux pratiques spécifiques aux tests, qui sont « tester » et « intégration continue ». La première signale le fait que XP suit un développement logiciel dit de « test-driven development » [2]. L'« intégration continue » dit que chaque composant doît être intégré

[2] les tests des composants sont progressivement écrits par les développeurs et les clients avant et pendant le

et passé des tests. Ensuite, ces composants doivent passer les tests d'acceptation du ou des utilisateurs/clients.

La méthode décrit un processus de tests et des pratiques pour les tests unitaires, d'intégration et de système.

4.3 Scrum

4.3.1 Présentation générale

La méthode Scrum apparaît en 1986 au Japon [HA04]. Scrum est un terme de rugby qui signifie « getting an out-of play ball into the game with team work ». C'est une méthode de développement adaptable, rapide mais également une méthode d'organisation.

Elle ne définit pas de technique de développement particulière mais se focalise sur le fonctionnement de l'équipe et assure la flexibilité du système dans un contexte très évolutif. Elle propose surtout des techniques et des pratiques de gestion.

Scrum comprend trois phases qui sont l'avant-jeu (le planning et l'architecture), la phase de jeu (assure la flexibilité) et l'après-jeu (la version)(voir la figure 4.2 - La méthode SCRUM).

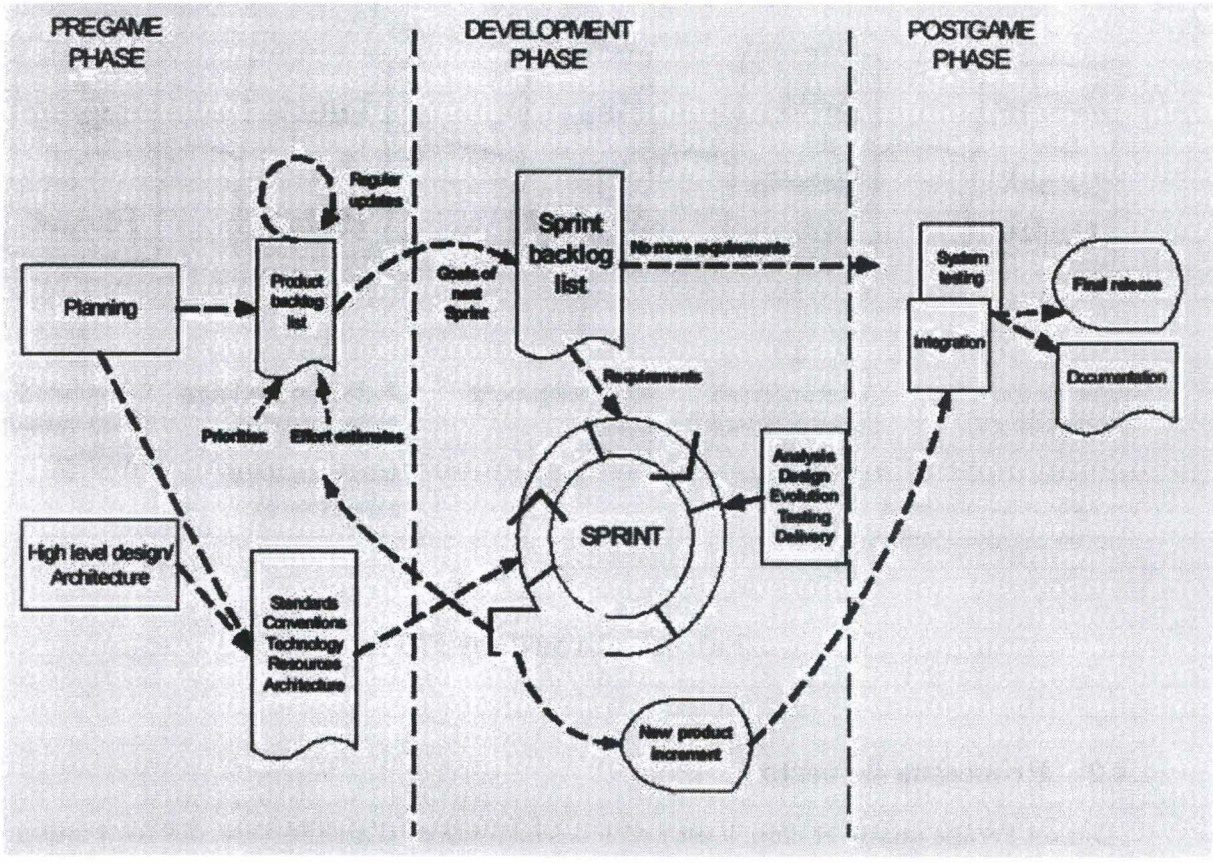


FIG. 4.2 – La méthode SCRUM

4.3.2 Processus de tests

Des tests d'intégration des différents composants sont prévus pendant la phase de jeu et des tests du système pendant l'après-jeu.

---

codage

Scrum [ASRW02] prévoit une gestion de projet pour les tests unitaires et d'intégration. Il est décrit aussi un processus de tests et des pratiques pour les tests système.

4.4 Feature Driven Development

4.4.1 Présentation générale

La méthode de développement FDD (Feauture Driven Development) a été créée par Palmer et Felsing au début des années '90 pour les services bancaires [HA04]. Cette méthode ne couvre pas tous les aspects du processus de développement, mais se focalise sur l'architecture du produit.

Elle préconise un développement itératif et décrit des bonnes pratiques. Elle est constituée de cinq étapes séquencielles qui sont le développement d'un modèle général, la construction d'une liste de fonctionnalités, le plan par fonctionnalité, l'architecture par fonctionnalité et la construction de la fonctionnalité (voir la figure 4.3 - La méthode FDD).

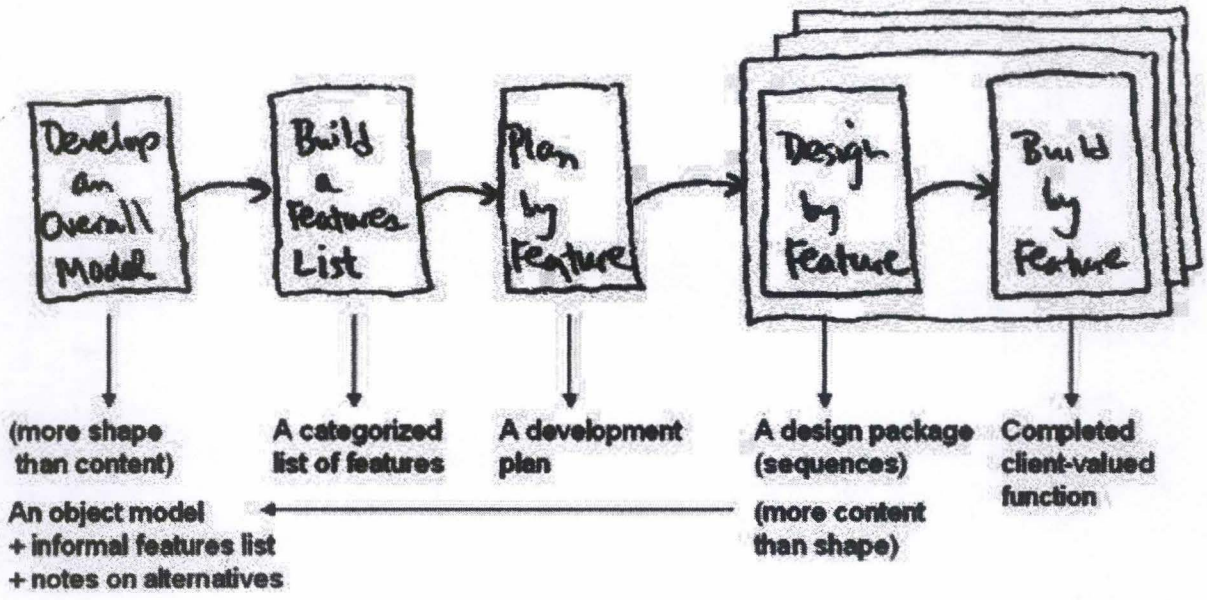


FIG. 4.3 – La méthode FDD

4.4.2 Processus de tests

Durant l'étape quatre et cinq, il est prévu des inspections d'architecture, des tests unitaires, des tests d'intégration et des inspections du code.

On trouve trois rôles [ASRW02] spécifiques aux tests, qui sont le testeur, le propriétaire de la classe et l'administrateur système. Le premier vérifie que le système produit rencontre les exigences de l'utilisateur. Il peut être indépendant de l'équipe de développement. Le propriétaire de la classe doit coder, tester et documenter la classe qu'il a développée. L'admistrateur système doit tester et évaluer l'environnement qui va être utilisé par l'équipe de développement.

Une pratique spécifique aux tests est l'inspection. Cette pratique est un mécanisme de détection de défauts dans le code ou l'architecture.

Enfin, FDD prévoit une gestion de projet, un processus de tests et des pratiques pour les tests unitaires, d'intégration et de système.

4.5 Dynamic Systems Development Method

4.5.1 Présentation générale

La méthode DSDM (Dynamic Sytems Development Method) a été élaboré en 1994 [HA04]. Cette méthode est itérative mais non rapide, découpe le projet en sous-projets et constitue un cadre de travail pour contrôler le RAD.

Contrairement aux autres méthodes de développement, DSDM estime le temps et les ressources disponibles et ensuite voit ce qui peut être développé. Elle est composé de cinq phases qui sont une étude de faisabilité, une étude Business, une itération fonctionnelle, une itération d'architecture et de construction, et une phase d'implémentation. Les deux premières phases sont séquentielles et faites une seule fois, et les trois dernières sont itératives et incréméntales (voir la figure 4.4 - La méthode DSDM).

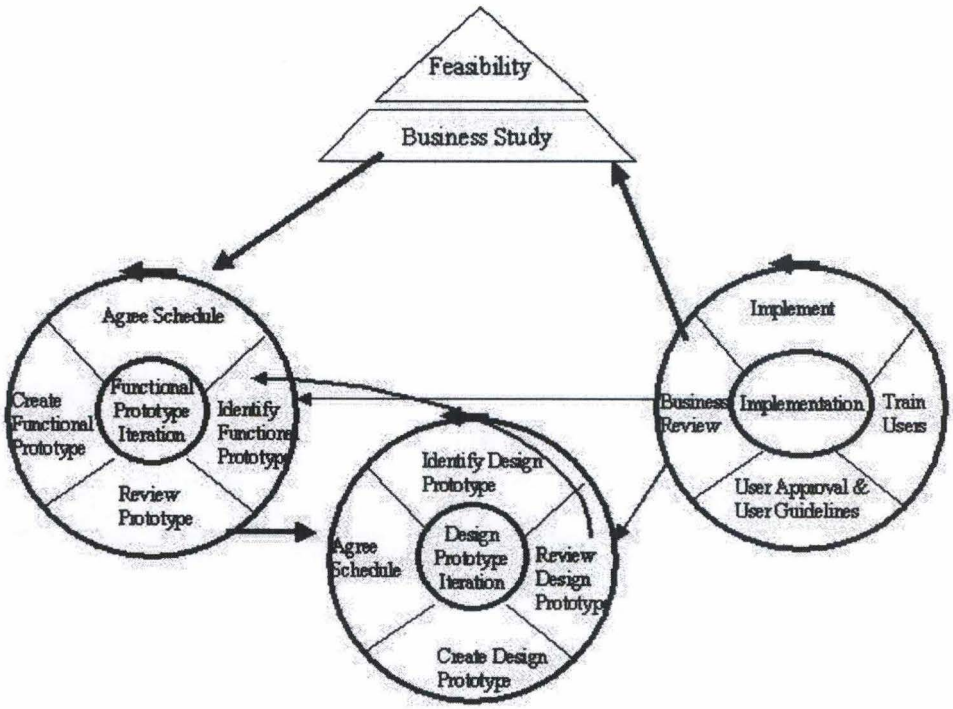


Figure 1: The DSDM Life-Cycle

FIG. 4.4 – La méthode DSDM

4.5.2 Processus de tests

On trouve un rôle [ASRW02] qui est le développeur. Le développeur doit tester.

On y trouve aussi une pratique. Cette pratique dit que les tests doivent être intégrés dans le cycle de vie du développement du produit, que les développeurs doivent faire des tests unitaires et, enfin, procéder à des tests de non-régression.

La méthode prévoit une gestion de projet et décrit un processus de tests pour les tests unitaires, d'intégration, de système et d'acceptation.

4.6 Rational Unified Process

4.6.1 Présentation générale

Le Rational Unified Process (RUP) est un processus de génie logiciel développé et commercialisé par Rational, qui a été racheté ensuite par IBM. Cette méthode a été proposée par Philippe Kruchten et Ivar Jacobsen, le père des cas d'utilisation, qui voulaient fournir une méthode supportant UML. L'objectif du processus est de produire des logiciels de grande qualité qui satisfassent les exigences des utilisateurs finaux, pour un coût et dans des délais prévisibles. RUP se subdivise en quatre phases (Inception, élaboration, construction et transition), qui sont divisées en itérations, chacune produisant une partie fonctionnelle du logiciel développé.

C'est un processus qui décrit qui fait quoi, comment et quand. Le membre est le « qui », c'est-à-dire les personnes affectées au projet. L'artéfact est le « quoi », c'est-à-dire les fichiers et les documents. L'activité est le « comment », c'est-à-dire les activités et les tâches. L'enchaînement d'activité est le « quand ».

Le modèle compte des dizaines de pratiques et quinze enchaînements d'activités dont six enchaînements d'activités d'ingénierie [Cae03].

L'approche itérative du RUP permet de diviser le cycle de vie d'un grand projet en une succession de petits projets en cascade. Il définit des jalons et des phases pour mesurer l'avancement du travail. Dans un développement itératif, les risques sont évalués au préalable. Les premières itérations permettent d'avoir des retours utilisateurs, le test et l'intégration sont continus, les jalons permettent de fixer les objectifs, les avancées sont mesurées au fur et à mesure de l'implémentation et des maquettes intermédiaires peuvent être déployées (Voir la figure 4.5 - La méthode RUP).

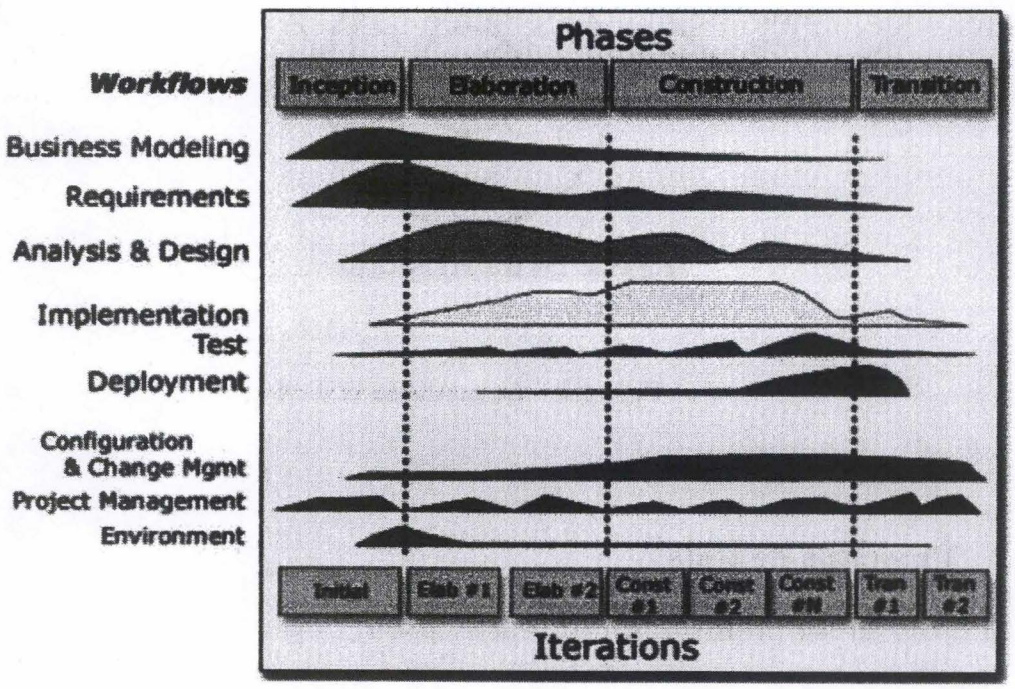


FIG. 4.5 – La méthode RUP

4.6.2 Processus de tests

Le cycle de vie défini par RUP a été vu dans le point 2.1.1 Exemple d'un cycle de vie des tests logiciels : RUP [Rat01].

Les types de tests ont été vus dans le point 1.2.4 Une vue des tests par dimensions.

Le Rational Unified Process [Rat01] distingue deux rôles spécifiques aux tests qui sont le concepteur des tests et le testeur. Le concepteur des tests a un rôle principal dans le déroulement de ceux-ci. Il est responsable de la planification, de la conception, de l'implémentation et de l'évaluation des tests. Il génère le plan et le modèle des tests, implémente les procédures de tests, évalue la couverture et l'efficacité des tests et génère le résumé de l'évaluation des tests. Il doit avoir la connaissance et l'expérience du système ainsi que des applications sous tests, des tests en général, des outils de tests automatiques et, du diagnostic et expérience des résolutions de problème. Le testeur est responsable de la configuration des tests, de l'exécution et de l'évaluation d'exécution des tests et de la reprise d'anomalie. Il y a deux types de testeur dit senior et junior. Généralement, on confie les tests de performance et d'intégration aux seniors de par leur expérience et les tests fonctionnels et de système aux juniors. Tous deux doivent avoir des connaissances dans les systèmes et applications sous test, dans les tests et outils de test automatique, dans le diagnostic et expérience dans la résolution de problème et, en programmation. Les seniors doivent aussi avoir de l'expérience des réseaux et architectures systèmes.

La méthode prévoit une gestion de projet, décrit un processus de tests et des pratiques pour les tests unitaires, d'intégration, de système et d'acceptation [ASRW02].

4.7 Le modèle Capability Maturity Model

4.7.1 Présentation générale

Elaboré en 1987 par Watts Humphrey, du Software Engineering Institute (SEI) de l'université Carnegie Mellon de Pittsburgh, le Capability Maturity Model (CMM) est un modèle d'évaluation et d'évolution des processus logiciels. Le modèle comporte cinq niveaux de maturité [CMM94] qui sont : initial, reproductible, défini, maîtrisé et optimisé. Ces niveaux représentent le cheminement par étape de l'entreprise pour arriver à des processus matures, c'est-à-dire conformes à un ensemble de bonnes pratiques observées à travers le monde dans des entreprises réputées pour bien gérer leurs processus. Plus on grimpe dans les niveaux de maturités, plus la capacité des processus augmente. Le CMM est structuré en processus clefs, nommés secteurs clefs, définissant ainsi un certain nombre d'activités à mettre en oeuvre, et de dispositions à prendre, mais laissant toute la latitude quant à la façon de les mettre en oeuvre.

4.7.2 Activités de tests

Les activités concernant les tests logiciels sont définies dans le niveau de maturité trois du CMM et plus précisément dans les activités réalisées cinq à sept du secteur clef « Ingénierie de produits logiciels ».

L'activité 5

« *Les tests de logiciel sont effectués conformément au processus logiciel défini du projet.* » [CMM94]

Les bonnes pratiques sont :

- Les critères de tests sont définis et passés en revue avec le client ou les utilisateurs finaux, selon le cas.

- Des méthodes efficaces sont utilisées pour tester le logiciel.
- La pertinence des tests est déterminée en fonction du niveau des tests effectués, de la stratégie d'essai choisi, et de la couverture voulue des tests.
- Pour chaque niveau de test logiciel, des critères de déclenchement des tests sont définis et appliqués.
- Des tests de non-régression sont effectués au besoin, à chaque niveau de test, quand des changements sont apportés au logiciel testé ou à l'environnement de ce dernier.
- Le plan, les procédures et les jeux de tests sont soumis à une revue par les pairs avant d'être considérés comme prêts à l'utilisation.
- Les plans, procédures et jeux de tests sont gérés et contrôlés.
- Les plans, procédures et jeux de tests sont modifiés comme il convient quand des changements sont apportés aux exigences allouées, aux exigences logicielles, à la conception du logiciel ou au code en cours de tests.

En ce qui concerne les tests possibles, on trouve les tests unitaires, les tests d'intégration, les tests système et les tests d'acceptation. Les types de stratégies de test comprennent les tests fonctionnels (appelé aussi boîte noire), les tests structurels (appelé aussi boîte blanche) et l'utilisation de statistiques.

L'activité 6

« *Les tests d'intégration du logiciel sont planifiés et affectées conformément au processus logiciel défini du projet.* » [CMM94]

Les bonnes pratiques sont :

- Les plans de tests d'intégration sont documentés et fondés sur le plan de développement logiciel.
- Les jeux de tests et procédures de tests d'intégration sont passés en revue avec les personnes responsables des exigences logicielles, de la conception du logiciel ainsi que des tests système et tests d'acceptation.
- Les tests d'intégration du logiciel sont effectués par rapport à la version spécifiée du document des exigences logicielles et du document de conception logiciel.

L'activité 7

« *Les tests système et tests d'acceptation logiciel sont planifiés et effectués afin de démontrer que le logiciel satisfait aux exigences énoncées.* » [CMM94]

Les bonnes pratiques sont :

- Les ressources en matière de test logiciel sont affectées suffisamment tôt pour que les tests puissent être préparés de façon adéquate.
- Les tests système et d'acceptation sont documentés dans un plan de tests qui est ensuite passé en revue avec le client et les utilisateurs finaux, selon le cas, et approuvé par ceux-ci.
- Les jeux de tests et procédures de tests sont planifiés et préparés par un groupe de test qui est indépendant des développeurs.
- Les jeux de tests sont documentés et passés en revue avec le client et les utilisateurs finaux, selon le cas, et approuvés par ceux-ci avant le début des tests.
- Les tests du logiciel sont effectués vis-à-vis du logiciel de référence des exigences allouées et exigences logiciel.
- Les problèmes identifiés au cours des tests sont documentés et font l'objet d'un suivi jusqu'à leur résolution.

- Les résultats des tests sont documentés et utilisés comme fondement pour déterminer si le logiciel satisfait aux exigences énoncées. Les résultats des tests sont gérés et contrôlés.

4.8 Le modèle ISO/SPICE ou norme ISO/IEC 15504

4.8.1 Présentation générale

Le projet SPICE, Software Process Improvement and Capability dEtermination devenu la norme ISO/IEC 15504en 1998, a pour objectif de proposer une modélisation pour les activités d'évaluation. Historiquement, SPICE s'est rapproché très tôt du modèle CMM pour devenir un méta-modèle. Il intègre un cahier des charges pour les modèles de maturité et propose une méthode d'évaluation des modèles. Contrairement au modèle CMM qui prévoit une évolution niveau par niveau, le modèle SPICE propose une évolution pratique par pratique, selon les besoins de l'organisation [fra98].

Le modèle est composé de deux dimensions. La première est la dimension "processus" qui identifie une quarantaine d'activités majeures. La deuxième est la dimension "aptitude" qui propose des modalités génériques de mise en oeuvre et de gestion de ces processus, selon une hiérarchie décrite en termes de niveaux d'aptitudes.

SPICE représente une structure formalisée et normalisée dédiée à la gestion des exigences d'un processus de développement logiciel. Ce modèle peut être utilisé par les organisations soucieuses de maîtriser leurs procédés de planification et de pilotage ainsi que d'améliorer les procédures d'acquisition. Il examine les procédés des organisations en matière de production du logiciel, puis détermine à quel niveau de maturité elles se situent. Le résultat de cette analyse en termes de risques et de faiblesses est ensuite utilisé pour orienter le processus d'amélioration.

Il y a six niveaux de maturité (Non effectué, effectué de façon informelle, planifié et suivi, bien défini, maîtrisé quantitativement et en amélioration permanente) et la dimension processus se décompose en cinq catégories de préoccupations (client-fournisseur, ingénierie, projet, support et organisation).

4.8.2 Processus de tests

Les tests dans ce modèle se composent de trois processus qui sont le processus d'intégration du logiciel, le processus de test du logiciel et le processus d'intégration et de test du système.

Processus d'intégration du logiciel

« *La finalité du processus d'intégration du logiciel est d'assembler les modules de logiciel en produisant des éléments intégrés de logiciel, et de vérifier que les modules de logiciel une fois intégrés reflètent proprement la conception du logiciel.* » [fra98]

Les pratiques de base de SPICE sont les suivantes :

- Développer une stratégie d'intégration des composants de logiciel cohérente avec la stratégie de livraison. Identifier les ensembles des composants de logiciel et une séquence ou un ordre pour les vérifier.
- Développer une stratégie de ré-exécution des tests des éléments de logiciel intégrés si un changement a été introduit dans un composant de logiciel.
- Décrire les tests qui doivent être exécutés pour chaque élément de logiciel intégré, en précisant les exigences du logiciel qui doivent être contrôlées et les critères de vérification.

- Vérifier chaque élément de logiciel intégré en utilisant les critères d'acceptation, et documenter les résultats.
- Intégrer les ensembles d'éléments de logiciel pour former un système logiciel complet. Assurer la cohérence entre les exigences du logiciel et la conception du logiciel.
- Si des changements sont apportés aux composants de logiciel, exécuter les tests de non-régression tel que définis dans la stratégie de test de non-régression.

On voit clairement que ce processus indique de pratiquer un test d'intégration avec une des méthodes vues au point 1.3.2 Les tests d'intégration. S'il y a modification quelconque, il est utile de faire un test de non-régression vu au point 1.3.4 Les tests de performances. Il faut mettre sur pied une stratégie d'intégration des composants identifiés et une stratégie de non-régression si un composant venait à changer. Il faut que chaque élément du logiciel soit testé. Puis il faut intégrer tous les éléments ensemble pour obtenir un logiciel intégré.

Processus de tests du logiciel

« *La finalité du processus d'essais du logiciel est d'essayer le logiciel intégré en produisant un produit qui satisfera aux exigences du logiciel.* » [fra98]

Les pratiques de base préconisées par SPICE sont les suivantes :

- Développer une stratégie de test du logiciel, et de ré-exécution des tests si un changement a été introduit dans un composant de logiciel.
- Décrire les tests à exécuter pour le produit logiciel complet, en indiquant les exigences logiciels à contrôler, les données en entrées et les critères d'acceptation. L'ensemble des essais doit démontrer la conformité aux exigences du logiciel.
- Vérifier le logiciel intégré en utilisant les critères d'acceptation et de documenter les résultats.
- Si des changements sont apportés aux composants de logiciel, exécuter les tests de non-régression tels que définis dans la stratégie de test de non-régression.

La réalisation des tests fonctionnels, vus au point 1.3.3 Les tests fonctionnels, va permettre de vérifier que le logiciel est conforme aux exigences. Les tests de performance, vus au point 1.3.4 Les tests de performances, sont aussi conseillés. Il faut développer une stratégie de test du logiciel et une stratégie de non-régression, définir les exigences à contrôler et effectuer les tests sur le logiciel complet, et documenter les résultats. Les tests de non-régression doivent être effectués si un changement intervient dans le logiciel.

Processus d'intégration et de test système

« *La finalité du processus d'intégration et d'essai du système est d'intégrer les composants de logiciel aux autres composants, comme les opérations manuelles ou le matériel, en produisant un système complet qui satisfera aux attentes des clients exprimées par des exigences du système. Il convient que les ressources impliquées dans l'intégration du système incluent une personne familière des composants de logiciel.* » [fra98]

Les pratiques de base préconisées par SPICE sont les suivantes :

- Développer une stratégie d'intégration des modules de système et d'essai du système cohérent avec la stratégie de livraison.
- Développer une stratégie de ré-exécution des essais des éléments si un changement a été introduit dans un module de système.

- Identifier les ensembles de composants de système et une séquence ou un ordre pour les tests.
- Décrire les tests à réaliser pour chaque ensemble de système, en indiquant les exigences à contrôler, les données en entrée, les éléments de système nécessaires pour réaliser les tests et les critères d'acceptation.
- Vérifier chaque ensemble de système pour s'assurer qu'il satisfait aux exigences, et documenter les résultats.
- Décrire les tests à réaliser pour le système intégré, en indiquant les exigences du système à contrôler, les données en entrée et les critères d'acceptation.
- Vérifier le système intégré pour s'assurer qu'il satisfait aux exigences du système, et documenter les résultats.
- Si des changements sont apportés dans les éléments du système, exécuter les tests de non-régression tels que définis dans la stratégie de test de non-régression.

Ce processus intègre les deux processus cités auparavant, en reprennent leurs procédés, en se focalisant plus sur les composants du logiciel. En plus, il faut développer une stratégie de test de non-régression si un composant change dans le logiciel et réaliser des jeux de tests sur l'ensemble des composants du logiciel et documenter les résultats.

4.9 Le modèle CMMI

4.9.1 Présentation générale

Pour résoudre le problème du nombre de modèles dérivés du SW-CMM (SE-CMM, PEOPLE-CMM ...), une intégration des différents modèles a fait apparaître le modèle CMMI, I pour « Integrated ». Dans ce modèle, le système et le logiciel sont intégrés. Commandité par le Département de la Défense des Etats-Unis et géré par le SEI, l'objectif de CMMI est l'élaboration d'un modèle intégrant les processus système et logiciel ainsi qu'une méthode d'évaluation dénommée SCAMPI, Standard CMMI Appraisal Method for Process Improvment [CMM02]. Il établit donc un lien entre les aspects système et logiciel, assure la maîtrise des coûts et des délais, améliore les performances des applications et systèmes développés. [Ama03]

Il existe deux représentations du modèle, connues sous le nom de représentation en continu (Continuous) et représentation en étape (Staged). Ces modèles sont structurés selon leurs prédécesseurs XX-CMM. Il y a deux représentations car il est difficile de définir une seule approche. Ce compromis dans le choix de deux représentations facilite la transition éventuelle entre CMM et CMMI.

La représentation en continu est basée sur deux dimensions : la dimension processus (Que faire ?) et capacité (Comment bien le faire ?) et suit la même philosophie que SPICE c'est-à-dire une amélioration par processus individuellement, selon les objectifs stratégiques de l'entreprise.

La représentation en étape comprend cinq niveaux de maturité qui sont initial, géré, défini, maîtrisé quantativement et optimisé. Pour le modèle CMMI en étape, chaque niveau de maturité est composé de secteurs clefs qui possèdent des objectifs guidés par les pratiques. Un niveau de maturité ne peut être atteint que si toutes les pratiques de l'entreprise concernée pour ce niveau satisfassent les exigences de ce niveau.

4.9.2 Processus de vérification des travaux produits

Le processus de vérification est mis en oeuvre par trois pratiques [CMM02]. La première est la préparation de la vérification. Dans cette pratique, il y a plusieurs démarches :

- Sélection des travaux produits à vérifier. Il faut identifier les produits à vérifier, identifier les exigences qui doivent être satisfaites par le produit, identifier les méthodes de vérification à utiliser, définir les méthodes de vérification qui sont utilisées pour chaque produit et enfin soumettre pour intégration avec le plan du projet l'identification du produit, les exigences à satisfaire et les méthodes utilisées. Pour les tests à utiliser, le modèle n'en impose pas mais en cite quelques uns comme les tests de chargement, de surcharge, de performance, fonctionnels, et d'acceptation mais aussi l'utilisation des jeux de tests.
- Etablir l'environnement des vérifications. Il faut identifier les exigences de l'environnement de vérification, identifier les ressources de vérification qui sont disponibles pour la réutilisation et les modifications, identifier les équipements et outils de vérification, acquérir des supports de vérification d'équipement et d'environnement, tel qu'un équipement et logiciel de test.
- Etablir des procédures et des critères de vérification. Il faut générer un ensemble de procédures de vérification compréhensible et intégré pour les produits, développer et raffiner le critère de vérification si nécessaire, identifier les résultats attendus, quelques tolérances autorisées dans l'observation, et d'autres critères pour satisfaire les exigences et identifier quelques équipements et composants environnementaux nécessaires pour mettre en oeuvre la vérification.

La deuxième pratique est l'exécution des revues. Dans cette pratique, il y a les démarches suivantes :

- Préparer des revues des travaux produits sélectionnés.
- Mener les revues sur les travaux produits sélectionnés et identifier les problèmes résultant de la revue.
- Analyser les données de la préparation, de la conduite et les résultats des revues.

La troisième pratique est la vérification des travaux produits sélectionnés. Cette pratique contient les démarches suivantes :

- Exécuter la vérification sur les travaux produits sélectionnés. Il faut exécuter la vérification des travaux produits sélectionnés par rapport à leurs exigences, enregistrer les résultats des activités de vérification, identifier les actions résultant de la vérification des travaux produits, documenter la méthode de vérification exécutée et les déviations des méthodes disponibles et procédures découvertes durant cette exécution.
- Analyser les résultats de toutes les activités de vérification et identifier les actions correctives. Il faut comparer les résultats obtenus avec les résultats attendus, identifier les produits qui ne satisfont pas à leurs exigences ou identifier les problèmes avec les méthodes, procédures, critères ... Puis, il faut analyser les défauts via la vérification des données, enregistrer tous les résultats de l'analyse dans un rapport, utiliser la vérification des résultats pour comparer les mesures réelles et performance avec les paramètres techniques de performance. Et enfin, fournir des informations sur la façon dont les défauts peuvent être résolus et la formaliser dans un plan.

Le modèle recommande aussi d'établir et de maintenir une organisation de règles d'actions pour la planification, le plan d'exécution et l'exécution des processus de vérification. Il faut fournir les ressources adéquates pour l'exécution des processus de vérification, développer les travaux produits et fournir les services des processus. Par exemple, les simulateurs, les générateurs de jeux de tests ... Il faut surveiller et contrôler les processus de vérification par rapport au plan pour exécuter les processus et prendre les actions correctives appropriées. Aucun test en particulier n'est imposé par le modèle, il faut juste mettre en oeuvre la bonne pratique.

4.10 Conclusion

Les méthodologies de développement et les modèles de qualité des logiciels donnent des indications sur les enchaînements des activités à effectuer ainsi que les tâches que chaque personne (rôles) de l'équipe de tests doit respecter. Elles définissent aussi les entrées et les sorties, tels que les documents à remplir, les *« templates »* ou modèles de document, ou à créer pour laisser des preuves que la tâche (responsabilités) est bien exécutée. En plus de cela, elles fournissent des bonnes pratiques (surtout dans les modèles de qualité) qui sont observées dans les entreprises, à respecter ou à suivre pour que les tests soient bien exécutés.

Les méthodologies de développement et les modèles de qualité des logiciels définissent aussi selon les cas une gestion de projet associée et un processus de tests.

On peut dire que les méthodes de développement et les modèles de qualité servent de guide pour mener à bien le projet, que se soit pour le développement général des logiciels ou pour les tests à produire dans le projet. Les modèles de qualité demandent une adaptation au contexte de l'entreprise lorsqu'ils sont mis en oeuvre.

Les tests à effectuer varient en fonction de la méthode ou du modèle choisi, et doivent donc être adaptés en fonction du contexte dans lequel le projet s'inscrit (taille, durée, enjeu ...).

Deuxième partie

Modélisation d'un processus de tests

Introduction

Cette deuxième partie est une modélisation d'un processus de tests général défini pour le CITI, suite à la première partie et aux interviews effectuées durant mon stage sur place. Chaque projet doit prendre le processus de tests général et l'adapter en fonction du contexte et des besoin du projet (cf. le chapitre 7, page 63). Chaque projet du CITI doit instancier le modèle de processus pour avoir sa propre instance de processus de tests. Cela permet de répondre aux besoins du projet, en terme de tests, en fonction de son contexte.

Suite aux lectures et aux interviews effectuées pendant ma période de stage au CITI, j'ai choisi une classification des tests par niveau de manière itérative. Pour chaque niveau de test, l'équipe projet décide s'il y a lieu de faire des tests de performance, fonctionnels ... En effet, c'est une classification qui est bien adaptée pour le CITI (voir la figure 4.6 - Classification des tests par niveau) qui englobe des projets de nature et aux besoins très différents.

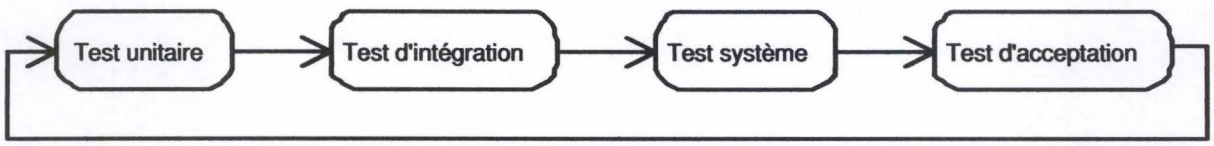


FIG. 4.6 – Classification des tests par niveau

On peut imaginer, par exemple, un développement en Java, dans lequel les tests unitaires se font sur les classes et la collaboration entre objets dans leur classe, les tests d'intégration sur les paquetages et l'interaction des classes à l'intérieur de leur paquetage et, enfin, les tests système sur les paquetages et les interactions entre eux.

Le processus de tests ainsi modélisé, prévoit une préparation des tests, décrit le déroulement des différents tests, les rôles et les responsabilités des personnes impliquées dans le projet, les activités et les tâches à effectuer, les délivrables à produire et à fournir, et une brève description de l'aide que peut fournir le langage de modélisation UML pendant le processus de tests.

Le processus est itératif et incrémental car tous les projets rencontrés au CITI, durant mon stage, suivaient un développement itératif par incrémentation.

Les délivrables/templates développés pendant mon stage pour le CITI se trouvent à l'annexe B Modèle de document, page 107. On y trouve un modèle de plan de test, un modèle pour les cas de test, un récapitulatif des cas de test et un modèle de fiche de livraison. Les check listes développées se trouvent à l'annexe C Check listes, page 117. On y trouve une check liste pour le plan de test et une check liste pour les cas de test. Le guide de création des cas de test et la description de demande d'évolution se trouvent dans l'annexe D Guides, page 119.

Cette partie comporte deux chapitres. Le chapitre 5 se focalise sur le déroulement des tests logiciels, et traite d'UML ainsi que l'aide que les diagrammes peuvent fournir aux testeurs durant leurs activités et tâches de tests. Le chapitre 6 donne les principales activités, tâches et délivrables que l'on devrait trouver dans un processus de tests logiciel et la manière de les modéliser avec l'aide d'UML.

Chapitre 5

Déroulement des tests logiciels

Il y a très peu d'ouvrages ou d'articles qui parlent d'UML dans les tests logiciels. Selon Williams [Wil04], UML a oublié de s'occuper de la partie consacrée aux tests. Il donne une idée de ce que pourrait apporter UML comme aide, par ses diagrammes, dans le processus de tests. Pour lui, les testeurs devraient peut-être mieux créer des diagrammes propres aux tests.

Ce chapitre traite du déroulement des tests logiciels que l'on peut mettre en oeuvre dans un projet. Je vais aussi faire le même raisonnement que Williams pour voir quel diagramme pourrait être utilisé afin d'aider les testeurs dans leurs tâches selon ma proposition de classification des tests.

Les sections de ce chapitre vont passer les types de tests, vus au point 1.3 Proposition de classification des types de tests, en développant le déroulement de chaque test, avant de développer une mise en oeuvre des tests logiciels dans le projet.

5.1 Les tests unitaires

5.1.1 Le déroulement des tests unitaires

Le test unitaire consiste à valider tous les composants du logiciel pris séparément. Ce test comporte différentes phases. La première est la spécification du test. Dans cette phase, on produit la spécification du test à partir du document de conception détaillée. La deuxième est la conception du test. On doit effectuer un choix de la technique de test utilisée. On peut pratiquer soit une technique visant la fonctionnalité [1], soit une technique visant la structure du code [2]. La troisième est l'exécution du test. On effectue le test manuellement ou de manière automatisée. Il faut reconstituer l'environnement du composant. La dernière est l'évaluation du test. En fonction des critères d'arrêt définis dans les objectifs du test, il faut évaluer si les tests sont terminés ou pas. Ensuite, on procède à la rédaction des fiches d'anomalies et à la définition des corrections à apporter.

Les tests unitaires se passent en différentes étapes :

- Examen du code. On parle de revues de code [Pfl01]. Cette technique sert à trouver les fautes et non pas à les corriger. Elle est faite par le développeur et une équipe de testeurs formée de 3 ou 4 experts techniques qui sont soit développeur, soit concepteur ou encore superviseur du projet. Deux types de revues de code sont possibles, la révision et l'inspection. Pour la révision, le code et la documentation sont présentés à l'équipe de revue. L'atmosphère est informelle, se focalise sur le code et non pas sur le codeur.

[1] Technique de la boîte noire
[2] Technique de la boîte blanche

Dans l'inspection, c'est le même principe que la méthode de révision mais avec une atmosphère plus formelle sous forme de réunion. Ici l'équipe de revue vérifie le code et la documentation par rapport à une liste préparée de préoccupations. Même si les développeurs n'aiment pas l'idée de se faire inspecter leur code ces méthodes marchent très bien. Plus vite une faute est trouvée, plus il est facile de la corriger et moins la correction coûte cher. Une autre idée est l'échange de codes entre les différents développeurs.

- Prouver que le code est correct. Il faut établir que le code, qui a été revu par l'équipe de revue, est correct. Un programme est correct lorsqu'il implémente les fonctions et les données comme indiqué dans l'architecture et est interfacé proprement avec les autres composants.

- Tester les composants du programme. Ici, on utilise des jeux de test. Pour tester un composant, on choisit des données d'entrées et des conditions exécution, on autorise le composant à manipuler ces données, et on observe la sortie qu'il produit. Un jeu de test est un choix particulier de données d'entrées qui est utilisé pour tester le programme. On peut voir un test comme une collection de jeux de test.

Généralement, le test unitaire est fait directement par le développeur du composant.

5.1.2 UML et les tests unitaires

La cible d'un test unitaire est le code. C'est pour cela que les diagrammes de cas d'utilisation, d'objet, de classe, d'états-transitions et de collaboration ou de séquence peuvent servir de support aux tests.

Les diagrammes de cas d'utilisation peuvent servir dans la conception des tests unitaires, comme par exemple les cas de test, et représentent la fonction de chaque objet/classe.

On peut se servir du diagramme d'objets pour présenter les instances d'objets d'une classe à tester et mettre les liens sémantiques entre les différents objets. On peut aussi l'utiliser pour montrer un contexte, par exemple avant et après une interaction entre objets.

On utilise le diagramme de classes en se focalisant, dans ce cas, sur la classe qui participe à un cas d'utilisation. Pour représenter le contexte précis, le diagramme peut être instancié en diagrammes d'objets.

Le but du diagramme de collaboration est de vérifier si les interactions entre objets, qui sont des instances de classe, existent. On y précise les états des objets qui interagissent, selon le contexte d'une interaction. On peut vérifier si l'ordre et la condition d'envoi de messages entre objets sont respectés. Le diagramme de séquence permet de vérifier la collaboration entre objets d'un point de vue temporel.

On peut utiliser le diagramme d'états pour vérifier les changements d'état d'un objet, en réponse aux interactions avec d'autres objets. Ce diagramme est complémentaire au diagramme de collaboration.

5.2 Les tests d'intégration

5.2.1 Le déroulement des tests d'intégration

L'objectif des tests d'intégration est de construire par étapes successives l'assemblage des composants testés unitairement, formant l'application logicielle. La stratégie d'intégration détermine l'ordre dans lequel les composants doivent être intégrés. Il existe plusieurs approches [Som04] et [Kon04].

La première approche est dite ascendante. Chaque composant du plus bas niveau de la hiérarchie du système est d'abord testé individuellement. Après, on teste les composants avec

celui qui se trouve juste au niveau au-dessus. Le test est fini lorsque tous les composants sont testés en même temps. Les avantages de cette approche sont que les tests s'appuient directement sur les composants réels, que les composants de bas niveau sont les plus testés et que la définition des jeux de test est plus aisée. Les inconvénients sont que les problèmes majeurs risquent d'être détectés tardivement et la planification des étapes d'intégration risque d'être remise en cause par l'indisponibilité des composants.

La seconde approche est dite descendante. C'est l'approche inverse. Le composant du niveau le plus haut est testé tout seul, avec des bouchons lors d'appel vers un composant de plus bas niveau qui n'est pas encore disponible. Puis, le composant du niveau le plus haut est testé avec les composants qu'il appelle, et ainsi de suite. Cette méthode se finit quand tous les composants sont incorporés. Les avantages de cette approche sont la détection précoce des défauts d'architecture, une détection plus rapide des anomalies majeures et une meilleure appréciation de la date de fin d'intégration. Les inconvénients sont qu'un effort important de simulation des composants absents est nécessaire et que cette intégration ne s'appuie pas sur les composants réels, donc risque d'apparition d'anomalie lors du remplacement des bouchons.

La troisième approche est dite massive. Tous les composants sont intégrés en une seule étape. Cela permet une intégration plus rapide. Généralement, ce type de test se fait sur les petits projets ayant une criticité faible. Sur les projets plus importants, de nombreuses anomalies risquent d'apparaître simultanément, ce qui entraîne une difficulté de localisation des anomalies et une diminution de l'efficacité du test. On perd plus de temps dans la recherche de l'origine de l'anomalie.

La dernière approche est dite mixte. Cette approche combine une stratégie descendante avec une stratégie ascendante. Elle permet de suivre le planning de développement car les premiers composants terminés sont intégrés en premier. Elle permet aussi de prendre en compte le risque lié à un composant. Ici, les composants les plus critiques sont intégrés en premier.

Généralement, la personne qui doit effectuer ces tests doit faire partie du projet, pour connaître l'architecture mise en place, et avoir des connaissances d'architecture en général.

5.2.2 UML et les tests d'intégration

La cible est l'intégration des composants/paquetages. Les diagrammes de cas d'utilisation, de composant, de séquence et d'activité peuvent servir de support à ces tests.

Les diagrammes de cas d'utilisation peuvent servir dans la conception des tests d'intégration comme par exemple les cas de test et peuvent vérifier les liens entre les fonctionnalités.

Dans le cas des tests d'intégration, les diagrammes de classe permettent de vérifier si la structure de l'ensemble des classes qui composent un paquetage est bonne. On peut y vérifier aussi les associations entre classes pour exprimer les liens entre celles-ci.

Les diagrammes de composant permettent de vérifier si l'organisation des composants dans les sous-systèmes et leurs dépendances sont bonnes.

Les diagrammes de séquence, dans ce cas-ci, permettent de vérifier si les participants et leurs responsabilités, la séquence des messages, les activités et leur synchronisation, ainsi que les conditions d'exécution entre classes d'un paquetage sont respectés.

Les diagrammes d'activité permettent, dans les tests d'intégration, de vérifier le déroulement d'un cas d'utilisation, la synchronisation et les couloirs d'activité.

Pour les tests système, la cible est le système intégré. C'est pour cela que tous les diagrammes cités précédemment peuvent servir de support au test. Cela dépend des autres tests effectués.

5.3 Les tests fonctionnels

5.3.1 Le déroulement des tests fonctionnels

Les principaux objectifs des tests fonctionnels sont la conformité fonctionnelle sur base de spécifications, l'ergonomie et la détection des éventuelles carences de spécifications.

On procède en testant les fonctionnalités du logiciel, les modes d'utilisation nominaux c'est à dire l'enchaînement de fonctions, l'enchaînement des fonctionnalités et tester les cas d'erreur prévus dans la spécification via des messages d'erreur applicatifs.

Le test fonctionnel ignore la structure mais se focalise sur la fonctionnalité. Il est basé sur les exigences fonctionnelles du système. On teste une par une les fonctionnalités que le système devrait avoir selon la documentation des exigences fonctionnelles.

La personne ou les personnes assignées à cette tâche doivent avoir des connaissances du domaine d'application. Ou alors une personne du développement et une personne du domaine en question, par exemple l'utilisateur, avec quelques connaissances en informatique. Cette personne peut faire partie du projet ou être indépendant.

5.3.2 UML et les tests fonctionnels

La cible est la fonctionnalité. C'est pour cela que les diagrammes de séquence, d'états/transitions et de cas d'utilisation peuvent servir de support au test.

Les diagrammes d'états-transitions permettent de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs. Ils peuvent servir, lors de tests fonctionnels à vérifier que les transitions entre états et changements d'états bien respectées.

Les diagrammes de cas d'utilisation peuvent servir à vérifier que les fonctionnalités prévus par le système sont bien présentes.

Les diagrammes de séquences, qui servent à illustrer des cas d'utilisation, peuvent servir à vérifier si les interactions entre les fonctionnalités sont respectées.

5.4 Les tests de performance

5.4.1 Le déroulement des tests de performance

L'objectif est de vérifier la conformité du logiciel aux exigences non fonctionnelles. Les tests de performance impliquent tout le système.

On procède en testant le comportement du système avec des conditions anormales, par exemple des volumes de données plus grands que les besoins. On vérifie le séquençage et le temps de réponse des transactions, les accès multiples sur une même ressource et la sécurité du système.

Différents tests de performances peuvent être réalisés comme un test de surcharge, un test de volume, un test de non-régression, un test de séquençage, un test de qualité et un test de reprise qui ont tous été décris au point 1.3.4 Les tests de performances.

Pour ces tests, il faut avoir des connaissances matérielles et logicielles. On peut imaginer plusieurs personnes, du projet ou indépendants, des experts en matériel et en logiciel informatique.

5.4.2 UML et les tests de performance

La cible du test de performance est de vérifier si le système résiste à la montée en charge, si le temps de réponse du système est conforme aux spécifications non fonctionnelles, si la contention et la reprise du système ont bien été définies. C'est pour cela que le diagramme de séquence peut servir dans ce cas ci.

Le diagramme de séquence pourra mettre le doigt sur une imperfection au niveau de la structuration des traitements, ou des contraintes d'exécution mal définies ou encore sur une synchronisation défectueuse.

5.5 Les tests d'acceptation

5.5.1 Le déroulement des tests d'acceptation

L'objectif de ce test est de permettre aux utilisateurs de déterminer si le système est en accord avec leurs besoins et leurs attentes.

On peut entreprendre différents types de test d'acceptation comme :

- Des tests de performance. L'utilisateur prépare un ensemble de jeux de tests qui représente les conditions typiques sur lequel le système va devoir opérer.

- Un test pilote. Décrit au point 1.3.6 Les tests d'installation.

- Des tests de parallélisme. Ces tests vont permettre aux utilisateurs de bien faire la comparaison entre l'ancien et le nouveau système ou d'assurer une transition graduelle vers le nouveau système.

L'utilisateur va mener les tests et définir les cas à tester. En théorie, les tests d'acceptation sont écrits, dirigés et évalués par l'utilisateur, avec l'assistance des développeurs si l'utilisateur se pose des questions d'ordre technique. Cet utilisateur doit faire partie des employés impliqués dans la définition des exigences pour avoir les connaissances du domaine.

5.5.2 UML et les tests d'acceptation

Les tests d'acceptation sont effectués par les utilisateurs. Je ne propose pas de diagramme pour cause de problèmes de connaissance technique et d'expérience concernant UML de la part de ceux-ci.

5.6 Les tests d'installation

5.6.1 Le déroulement des tests d'installation

La dernière étape de test implique l'installation du système sur le site de l'utilisateur. Si le test d'acceptation a été fait sur le site de l'utilisateur, alors ce test n'est pas nécessaire. Par contre, il est utile si les conditions du test d'acceptation ne sont pas les même de celles du client. Un test de non-régression est effectué pour déterminer si le système fontionne proprement.

Le test se focalise sur deux points : l'installation totale du système et la vérification de quelques caractéristiques fonctionnelles et non fonctionnelles par rapport au site. Lorsque l'utilisateur est satisfait des résultats, les tests sont terminés et le système est formellement délivré.

Le test d'installation exige le travail des informaticiens, des connaissances matérielles et logicielles, pour travailler avec l'utilisateur et déterminer quels sont les tests à effectuer sur le site. On peut imaginer une ou plusieurs personnes s'occupant du matériel informatique à installer et d'autres qui s'occuperaient des logiciels à installer.

5.6.2 UML et les tests d'installation

La cible est l'installation du système sur le site de l'utilisateur/client. Cela implique le déploiement du logiciel sur ce site. C'est pour cela que les diagrammes de déploiement et de composants peuvent servir de support aux tests.

Le diagramme de déploiement peut servir à vérifier la disposition physique des matériels qui composent le système, ainsi que la répartition des composants sur ces matériels.

Les diagrammes de composants permettent de vérifier si l'architecture physique des composants d'une application est bien respectée.

5.7 Tableau récapitulatif des diagrammes proposés

Type de tests	**Diagramme(s) proposé(s)**
Test unitaire	Cas d'utilisation, d'objet, de classe, d'état-transition, de collaboration et de séquence.
Test d'intégration	Cas d'utilisation, de classe, de composant, de séquence et d'activité.
Test fonctionnel	Cas d'utilisation, de séquence et d'état-transition
Test de performance	De séquence
Test d'acceptation	-
Test d'installation	De composant et de déploiement

TAB. 5.1 – Récapitulatif des diagrammes UML par type de tests

5.8 Conclusion

Un point clé qui ressort de ce chapitre est l'importance des connaissances requises à chaque étape de test. Un développeur qui a de très bonne connaissance dans le matériel informatique et qui n'en a pas dans les logiciels doit être assigné aux tests d'installation, par exemple.

Un autre point à signaler est l'importance de l'engagement des personnes assignées aux tâches des tests. On voit clairement que, si la personne ne fait pas bien le test unitaire comme il le faut, les personnes s'occupant des tests d'intégration, vont se retrouver avec une liste impréssionnante d'anomalies et provoquer des retards dans le déroulement du projet. On peut encore aller plus loin, en disant que si toutes les personnes assignées aux tâches des tests ne font pas ce qu'elles doivent faire, c'est l'utilisateur final qui va se retrouver avec un logiciel inutilisable ...

J'ai développé une idée pour savoir quels diagrammes peuvent servir d'aide aux testeurs durant leurs tâches. Pour avoir une conclusion probante sur ce sujet, il faudrait mener une étude en prenant une équipe de testeurs qui devrait pratiquer les tests avec les diagrammes d'UML. Ensuite, mener une enquête en faisant des interviews et pouvoir dégager des propositions.

Chapitre 6

Mise en oeuvre des tests logiciels

Ce chapitre recense les supports disponibles, selon les lectures que j'ai effectuées, pour la mise en oeuvre de tests logiciels dans un projet informatique suite à la partie précédente. Je me suis inspiré, entre autres, des descriptions très détaillées dans le CD-Rom de Rational [Rat01]. Les deux grandes parties traitent des supports disponibles pour la préparation et la réalisation des tests logiciels. On y recense les activités à effectuer. Chaque activité est divisée en tâches ou en responsabilités qu'une personne doit effectuer en s'aidant des supports disponibles. Les supports en question peuvent êtres des délivrables en entrée ou en sortie de l'activité ou des outils. La section suivante est un récapitulatif de tous les délivrables rencontrés pouvant servir de support aux tests logiciels. On y trouve aussi un récapitulatif des rôles les plus rencontrés dans le processus de tests logiciels. Ensuite, une section est consacrée à la possibilité d'utiliser UML pour modéliser les activités, les tâches, les délivrables ... Et enfin, une dernière section consacrée à la prise en compte du contexte du projet.

6.1 Les principales activités dans le processus de tests

J'ai rassemblé les activités à effectuer pendant le processus de tests en deux sections : les activités qui ont trait à la préparation des tests et celles qui ont trait à la réalisation des tests (Voir la figure 6.1 - Le processus de tests).

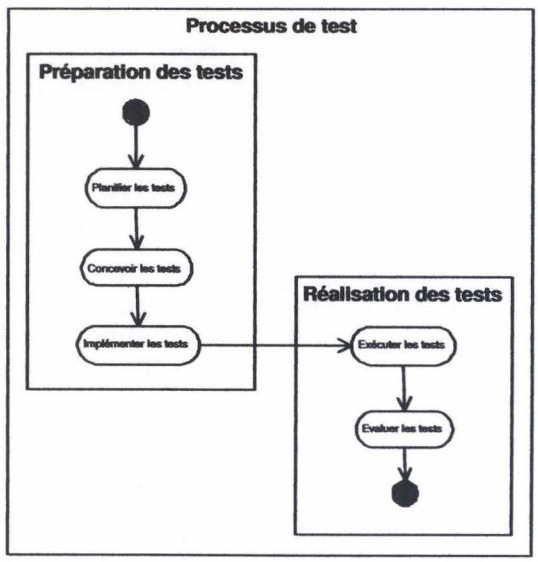


FIG. 6.1 – Le processus de tests

Pour la préparation des tests, j'ai distingué les activités de planification des tests, de conception des tests et d'implémentation des tests. Pour la réalisation des tests, j'ai distingué les activités d'exécution des tests et d'évaluation des tests. Les activités spécifiques à la préparation des tests logiciels sont présentées au point suivant et celles spécifiques à la réalisation des tests au point 6.3 Les activités concernant la réalisation des tests logiciels.

6.2 Les activités concernant la préparation des tests logiciels

6.2.1 L'activité de planification des tests

Cette activité a pour but de collecter et d'organiser l'information en vue de planifier les tests.

Les tâches de cette activité sont de :

- Repérer les exigences nécessaires pour les tests,
- Analyser les risques,
- Développer une stratégie de test,
- Identifier les compétences et les ressources nécessaires aux tests,
- Créer un planning des tests,
- Générer le plan de test.

Le responsable de cette activité est un concepteur des tests.

Les délivrables en entrée

Les spécifications supplémentaires

C'est un document présentant les exigences systèmes que les cas d'utilisation ne représentent pas. On peut trouver entre autres des exigences légales et réglementaires.

La personne qui rédige ce document est un analyste système. Le document se présente sous forme de description textuelle.

Le plan d'intégration

C'est un document fournissant un plan détaillé pour l'intégration des différents composants pour assembler le système. Il est défini l'ordre dans lequel les composants et les sous-systèmes vont être implémentés et dans quelle version exécutable ils vont être intégrés après l'intégration avec le système. La personne qui rédige ce document est un intégrateur.

Le guide de test

C'est une description/guide des tests. Il décrit comment définir la stratégie de test pour chaque itération.

La personne qui rédige ce document est un concepteur de test. Le document se présente sous forme de description textuelle.

Le planning

C'est un document représentant le séquençage dans le temps des activités et des tâches, avec les ressources assignées et contenant les dépendances des tâches. Ce document est appelé

plan d'itération dans le cas d'un développement de logiciel par itération, dans le RUP par exemple.

La personne qui rédige ce document est un chef de projet. Le document se présente sous forme de document Word, HTML, Microsoft Project ou d'autres outils de support de projet.

Le document d'architecture du logiciel

Ce document fournit une vue complète de l'architecture du système, utilisant plusieurs vues, une pour chaque aspect du système, par exemple, en formalisant avec UML comme RUP. La personne qui rédige ce document est un architecte/concepteur.

Le modèle de conception

C'est un modèle de classes décrivant les éléments statiques du système. Le modèle de conception est utilisé comme une donnée essentielle pour les activités dans l'implémentation et les tests. La personne qui rédige ce document est un architecte/concepteur. Le document se présente sous forme de stéréotype UML appelé « modèle de conception ».

Le modèle de composants

C'est un diagramme qui présente une collection de composants, et de sous-systèmes qu'ils contiennent. La personne qui rédige ce document est un architecte/concepteur.

Les documents en sortie

Le plan de test

Ce document contient les informations sur l'objectif et les buts des tests à effectuer. Il décrit les stratégies de test, l'exécution des tests, les ressources nécessaires pour effectuer les tests, les types de test à effectuer, la planification des tests et les délivrables à produire. Il est diffusé à l'ensemble de l'équipe de test. La personne qui rédige ce document est un concepteur de test.

Récapitulatif des délivrables

Voici un tableau reprenant les délivrables en entrée et en sortie de l'activité de planification des tests qui a pour but de créer le plan de test :

Entrée	Sortie
Spécifications supplémentaires	Plan de test
Modèle de conception	-
Modèle de composants	-
Document d'architecture du logiciel	-
Planning	-
Guide des tests	-
Plan d'intégration	-

TAB. 6.1 – Délivrables de support à la planification

6.2.2 L'activité de conception des tests

Cette activité a pour but d'identifier un ensemble de cas de test vérifiables pour chaque version exécutable et d'identifier les procédures de test qui montrent comment les tests sont réalisés.

Les tâches de l'activité sont de :

- Analyser la charge de travail (seulement pour les tests de performance),
- Identifier et décrire les cas de test,
- Identifier et décrire les procédures de test,
- Revoir et estimer la couverture des tests.

Le responsable de cette activité est un concepteur de test.

Les délivrables en entrée

Le cas d'utilisation

Ce document définit un ensemble d'instances de cas d'utilisation, où chacun est une séquence d'actions que le système accomplit et qui donne un résultat observable aux acteurs. Les cas d'utilisation sont des listes d'actions regroupées en vue de résoudre un problème donné. Ils définissent le service que doit fournir une application. La personne qui rédige ce document est un spécificateur de cas d'utilisation. Le document se présente sous forme de document Word ou HTML. La représentation UML est le cas d'utilisation avec le diagramme d'activité qui permet de décrire ses détails.

Les spécifications supplémentaires

Décrit au point 6.2.1 L'activité de planification des tests.

Le plan de test

Décrit au point 6.2.1 L'activité de planification des tests.

Les composants

Un composant, comme défini au point 1.2 Les tests logiciels, représente un bout de logiciel qui se conforme à un ensemble de services (source, binaire et exécutable), ou un fichier contenant l'information. Il peut aussi être un agrégat d'autres composants. La personne qui crée ces éléments est un développeur. La représentation UML se fait via les stéréotypes, comme par exemple, via un fichier exécutable.

Les délivrables en sortie

Le cas de test

Un cas de test décrit les conditions particulières de réalisation d'un test spécifique. Il présente les données d'entrée à appliquer à l'application testée, les conditions d'exécution, et les résultats attendus suite à l'application de ces données lors de la réalisation du test. La personne qui rédige ce document est un concepteur de test.

La procédure de test

C'est un ensemble d'instructions détaillées pour la configuration, l'exécution et l'évaluation des résultats pour un cas de test donné ou un ensemble de cas de test. La personne qui rédige ce document est un concepteur de test.

Le document d'analyse de la charge de travail

Ce document identifie les variables et définit leurs valeurs dans les différents tests de performance pour simuler et/ou émuler les caractéristiques de l'action, les fonctions métier des clients finaux, via les cas d'utilisation, la charge et le volume. La personne qui rédige ce document est un concepteur des tests.

Récapitulatif des délivrables

Voici un tableau reprenant les délivrables en entrée et en sortie de l'activité de conception des tests qui a pour but principal de créer les cas de test et les procédures de test :

Entrée	Sortie
Plan de test	Cas de test
Cas d'utilisation	Procédures de test
Spécifications supplémentaires	Document d'analyse de la charge de travail
Composants	-

TAB. 6.2 – Délivrables de support à la conception

6.2.3 L'activité d'implémentation des tests

Cette activité a pour but de créer ou de générer des scripts de test réutilisables et de garantir la traçabilité entre les différents éléments comme les cas de test et les cas d'utilisation.

Les tâches de l'activité sont de :

- Enregistrer, générer et programmer les scripts de test,
- Etablir l'ensemble des données dont les scripts auront besoin lors de leur exécution.

Le responsable de cette activité est un concepteur de test.

Les délivrables en entrée

Le document d'analyse de la charge de travail

Décrit au point 6.2.2 L'activité de conception des tests.

Les cas de test

Décrit au point 6.2.2 L'activité de conception des tests.

La procédure de test

Décrit au point 6.2.2 L'activité de conception des tests.

Les délivrables en sortie

Les scripts de test

Les scripts de test sont des instructions lisibles par l'ordinateur qui automatisent l'exécution d'une procédure de test (ou une portion). Ils peuvent être créés ou générés automatiquement en utilisant des outils de test automatiques, programmés en utilisant des langages de programmation, ou une combinaison d'enregistrement, de génération et de programmation. La personne qui rédige ce document est un concepteur des tests.

Récapitulatif des délivrables

Voici un tableau reprenant les délivrables en entrée et en sortie de l'activité d'implémentation des tests qui a pour but principal de créer des scripts de test :

Entrée	Sortie
Cas de test	Scripts de test
Procédures de test	-
Document d'analyse de la charge de travail	-

TAB. 6.3 – Délivrables de support à l'implémentation

6.3 Les activités concernant la réalisation des tests logiciels

6.3.1 L'activité d'exécution des tests

Cette activité a pour but l'exécution des tests et la collecte des résultats suite à l'exécution des tests.

Les tâches de l'activité sont de :

- Exécuter les procédures de test,
- Evaluer l'exécution des tests,
- Vérifier les résultats des tests,
- Reprendre les tests interrompus.

Le responsable de cette activité est un testeur.

Les délivrables en entrée

Les scripts de test

Décrit au point 6.2.3 L'activité d'implémentation des tests.

La version exécutable

Une version exécutable correspond à un ou plusieurs composants (parfois exécutables), chacun construit à partir d'autres composants, habituellement par un processus de compilation et de liaison de code source. La personne qui prépare cet élément est un intégrateur.

Les délivrables en sortie

Les résultats des tests

Une base de données doit contenir un répertoire de données capturées durant l'exécution des différents tests. Elle est utilisée dans le calcul de différentes mesures clefs des tests.

Récapitulatif des délivrables

Voici un tableau reprenant les documents en entrée et en sortie de l'activité d'exécution des tests qui a pour but principal de créer des documents sur les résultats des tests :

Entrée	Sortie
Scripts de test	Résultats des tests
Version exécutable	-

TAB. 6.4 – Délivrables de support à l'exécution

6.3.2 L'activité d'évaluation des tests

Cette activité a pour but l'évaluation des résultats des tests, la génération des requêtes de changement, le calcul des mesures clefs des tests et la génération du récapitulatif des résultats des tests.

Les tâches de l'activité sont de :

- Analyser les résultats et soumettre les requêtes de changement,
- Evaluer les exigences de base dans la couverture des tests,
- Analyser les défauts,
- Déterminer si l'achèvement des tests et le critère de succès sont atteints,
- Générer le récapitulatif de l'évaluation des tests.

Le responsable de cette phase est un concepteur de tests.

Les délivrables en entrée

Les résultats des tests

Décrit au point 6.3.1 L'activité d'exécution des tests.

Les délivrables en sortie

Les requêtes de changement

Les requêtes de changement sont utilisées pour documenter et tracer les défauts. Elles fournissent un enregistrement des défauts et des décisions prises lors du processus d'évaluation des

tests et assurent la diffusion des impacts de ces défauts. La personne qui rédige ce document est un concepteur des tests.

Le récapitulatif d'évaluation des tests

Ce document organise et présente les résultats du test et les mesures clefs pour une revue et une estimation. De plus, le document contient une évaluation par le testeur et le concepteur des tests, ainsi que des recommandations pour les futurs efforts. La personne qui rédige ce document est le concepteur de test. Le document se présente sous une description textuelle.

Récapitulatif des délivrables

Voici un tableau reprenant les délivrables en entrée et en sortie de l'activité d'évaluation des tests qui a pour but principal de créer des requêtes de changement et le récapitulatif d'évaluation des tests :

Entrée	Sortie
Résultats des tests	Requêtes de changement
-	Récapitulatif d'évaluation des tests

TAB. 6.5 – Délivrables de support à l'évaluation

6.4 Récapitulatif des supports disponibles pour les activités

6.4.1 Les délivrables

Délivrable/ support	Crée par l'activité	Utilisé par l'activité	Personne responsable
Spécifications supplémentaires	-	Planification/conception des tests	Analyste système
Plan d'intégration	-	Planification des tests	Intégrateur
Guide de test	-	Planification des tests	Concepteur de test
Planning	-	Planification des tests	Chef de projet
Document d'architecture du logiciel	-	Planification des tests	Concepteur
Modèle de conception	-	Planification des tests	Concepteur
Modèle de composants	-	Planification des tests	Concepteur
Plan de test	Planification des tests	Conception des tests	Concepteur de test
Cas d'utilisation	-	Conception des tests	Spécificateur
Cas de test	Conception des tests	Implémentation/exécution des tests	Concepteur de test
Procédure de test	Conception des tests	Implémentation/exécution des tests	Concepteur de test
Document d'analyse de charge travail	Conception des tests	Implémentation /exécution des tests	Concepteur de test
Script de test	Implémentation des tests	Exécution des tests	Concepteur de test
Résultats des tests	Exécution des tests	Evaluation des tests	Testeur
Version exécutable	-	Exécution des tests	Intégrateur
Requêtes de changement	Exécution des tests	-	Concepteur de test
Récapitulatif d' évaluation des tests	Exécution des tests	-	Concepteur de test
Requêtes de changement	Exécution des tests	-	Concepteur de test

TAB. 6.6 – Récapitulatif des supports

Les « - »,dans le tableau ci-dessus, signifient que le délivrable est soit créé ou soit utilisé par une activité spécifique au développement du logiciel.

6.4.2 La description des rôles

Rôle	Description
Analyste système	Cette personne est responsable de mener et de coordonner l'obtention des exigences et de les modéliser en cas d'utilisation, de définir les fonctionnalités et de délimiter le système. Par exemple, il doit établir quels acteurs et quels cas d'utilisation existent, et comment ils interagissent ensemble.
Architecte/ Concepteur	Cette personne est responsable de définir les responsabilités, les opérations, les attributs, les relations d'une ou plusieurs classes. Il est aussi responsable de la conception d'un ou plusieurs paquetages ou sous-systèmes.
Chef de projet	En général, le chef de projet est responsable de la conduite du projet : de la planification, de l'allocation des ressources, du suivi du projet. Au niveau des tests, il est responsable du planning des tests et de générer le plan de test à chaque nouvelle itération de test. Il est également responsable de la stratégie de test et de l'allocation des ressources pour les tests.
Concepteur de test	Cette personne est responsable de la conception, de l'implémentation et de l'évaluation des tests.
Développeur	Il est responsable du développement des composants et des tests de ceux-ci. Il est responsable de la pratique des tests unitaires sur ses composants et de la correction des défauts détectés lors de tous les tests.
Intégrateur	Il est responsable d'intégrer les composants en un sous-système pour les tests d'intégration et aussi d'intégrer le sous-système au système pour former le système intégré. Dans les projets de petite taille, ce rôle est tenu par les développeurs.
Relecteur	Il est responsable d'assurer la qualité du code source et de mener la revue du code source. Il est aussi responsable d'un retour sur le résultat de son activité de revue. Dans les projets de petite taille, ce rôle est tenu par les développeurs.
Spécifieur	Cette personne est responsable de détailler la spécification d'une partie des fonctionnalités du système en décrivant les exigences d'un ou plusieurs cas d'utilisation et d'autres exigences logicielles. Il est aussi responsable d' un paquetage de cas d'utilisation, et de maintenir l'intégrité de ce paquetage.
Testeur	Il est responsable des exécutions des tests d'intégration sur les sous-systèmes, et des tests système sur le système intégré. Il est responsable aussi de l'évaluation de ses tests.
Utilisateur	L'utilisateur est la personne qui va utiliser le système développé lorsque celui-ci sera livré. Il est responsable de la réalisation des tests d'acceptation.

TAB. 6.7 – Description des rôles

6.5 Proposition de modélisation du processus de tests avec UML

Cette proposition de modélisation du processus de tests a été développée avec la version 1.4 d'UML.

6.5.1 La modélisation du cycle de vie

Il n'y a pas de diagramme qui permette de bien modéliser le cycle de vie. Rien n'est prévu pour représenter une suite logique de phases dans le temps.

6.5.2 La modélisation des types de tests

Il n'y a pas de diagramme qui permette de bien modéliser les différents types de tests à effectuer.

6.5.3 La modélisation des rôles et des délivrables

Il n'y a rien de prévu dans UML pour représenter les rôles ou les délivrables et de leur assigner une définition ou une description.

6.5.4 La modélisation des activités et des tâches

Le diagramme d'activité est un bon outil pour modéliser une activité et les tâches qui lui sont attachées, ainsi que les délivrables en entrée ou en sortie de chaque tâche. Il est aussi possible de décrire les connexions entre activités, tâches et délivrables dans le temps.

Par contre, ce diagramme ne permet pas directement d'attacher un but/objectif à l'activité, de rajouter une description de chaque tâche et d'identifier la personne responsable de cette activité ainsi que l'outil utilisé.

6.5.5 La modélisation de l'état d'un délivrable

Le diagramme d'états est un bon outil pour modéliser les changements d'états d'un délivrable tout au long de son utilisation.

Par contre, ce diagramme ne permet pas directement d'identifier la personne responsable du délivrable ou de la personne responsable du changement d'état.

6.6 Le contexte du projet

Avant de choisir une méthodologie particulière, il faut faire une analyse du contexte du projet. Celle-ci peut s'effectuer, notamment, par les composantes et les facteurs critiques suivants.

6.6.1 Les quatre composantes

Le groupe DMR [eodsd87], définit quatre composantes (les usagers [1], les biens livrables, l'équipe de développement et l'environnement de développement) du contexte du projet associés à quatre attributs (la taille, la complexité, la spécialité et l'adéquation).

C'est-à-dire qu'il faut tenir compte du nombre d'usagers affectés par le projet, l'hétérogénéité des besoins des usagers et les premières expériences de mécanisation chez l'usager.

[1] Englobe le comité de direction du projet, l'équipe, les utilisateurs ...

Pour les biens livrables, il faut tenir compte de l'envergure de ceux-ci, par exemple le nombre de sous-systèmes de livraison, de la complexité de ceux-ci, par exemple le nombre d'unités de traitements complexes, la spécialisation du logiciel et l'entretien adapté.

L'envergure de l'équipe de développement est importante, c'est-à-dire le nombre de personnes impliquées, l'effort global de l'équipe et leurs besoins de communication.

L'environnement de développement n'est pas à négliger, c'est-à-dire l'introduction de nouveaux outils de développement et le nombre d'outils de développement à intégrer.

Il faut encore tenir compte de l'expérience de l'équipe de développement. Ont-ils déjà participé à des projets de même envergure, dans le même environnement de développement ou dans le même domaine d'application ?

6.6.2 Les cinq facteurs critiques

Boehm et Turner [HA04] ont distingué cinq facteurs critiques que l'on doit analyser avant tout choix méthodologique.

Le premier facteur est la taille de l'équipe. Si c'est un projet avec une petite équipe et une forte dépendance au mode de connaissance tacite [2], il vaut mieux choisir une méthode de développement comme XP, par exemple. Par contre, si c'est un projet avec une grande équipe et des règles très strictes, alors il vaut mieux utiliser un modèle plus élaboré comme CMM, par exemple.

Le deuxième facteur est la criticité du projet. S'il y a risque de pertes humaines (par exemple), il vaut mieux un modèle lourd comme CMM, par exemple. Sinon, pour les projets non critiques, on peut se contenter d'une méthodologie moins élaborée, comme XP.

Le troisième facteur est le dynamisme du projet. Si le taux d'exigences par mois varie beaucoup, il vaut mieux adopter une méthodologie comme RUP ou XP. Si par contre le projet évolue dans un environnement stable, on peut adopter un modèle comme CMM.

Le quatrième facteur est l'expérience du personnel. Les méthodes qui sont plus flexibles aux variations d'exigences demanderont un personnel plus expérimenté.

Le dernier facteur est la culture d'entreprise. Si l'entreprise a une organisation dite de « chaos », laissant une grande marge de liberté aux employés, une méthode moins lourde est adaptée. Par contre, si l'organisation de l'entreprise est dite d'« ordre », un modèle plus élaboré peut être plus facilement adapté.

[2] connaissance par expérience personnel non inscrit dans les documents mais partagé dans le goupe de manière informel

6.7 Conclusion

Ce chapitre donne donc une liste d'activités, de tâches et de délivrables qui peut être tantôt exhaustive et tantôt non exhaustive. Cela dépend de deux choses, la méthodologie de développement et/ou le modèle de qualité que l'entreprise a décidé de mettre en oeuvre ainsi que le contexte dans lequel le projet se déroule. Il est clair qu'une entreprise qui a adopté une culture d'entreprise autour de CMM, a une liste d'activités et de délivrables beaucoup plus grande et plus stricte. Il est clair aussi qu'un projet très peu critique constitué d'une poignée de développeurs avec un budget limité, n'adoptera pas le processus de tests de RUP. Le contexte du projet doit être pris en compte avant d'adopter une méthode ou un modèle.

Les diagrammes d'UML permettent de modéliser les activités et leurs tâches, leurs délivrables en entrée et en sortie, leurs interconnexions dans le temps (séquence), et de montrer l'état d'un délivrable dans le temps. Par contre, les diagrammes d'UML ne permettent pas directement d'identifier la personne responsable, de marquer le but/objectif atteint par l'activité ni de mettre des descriptions sur les différents éléments représentés. En général, les diagrammes ne représentent pas une bonne modélisation pour le cycle de vie, les types de tests à effectuer et, la représentation des rôles et des délivrables.

Troisième partie

Etude de cas

Chapitre 7

Démarche de tests logiciels pour le CITI

7.1 Introduction

Suite aux entretiens menés au CITI pour déterminer le degré de maturité du processus de tests logiciel (voir l'annexe A Questionnaire et évaluation, page 91), il a été constaté qu'il n'y avait aucune stratégie ni cycle de vie de test développé dans la majorité des projets. On ne sait pas quel type de test est pratiqué à quelle étape. Et il n'y avait pas non plus de guide méthodologique à disposition des testeurs, c'est-à-dire aucun standard, aucune uniformisation des techniques de test au CITI. La documentation est elle-même négligée. Les tests ne sont pas planifiés à l'avance et certains tests sont pratiqués en urgence. Tout ceci entraîne des délais trop courts pour tester le logiciel. En conséquence, les tests sont réalisés sans méthode, quand l'équipe a le temps et de façon différente selon les individus. Il faut aussi signaler que quelques projets avaient commencé une méthode XP et RAD mais l'on abandonné.

Le but de la démarche de tests qui est présentée dans ce chapitre, est de combler les manques dans la pratique des tests logiciels au sein du CITI. La démarche décrit un cycle de vie pour les tests logiciels, définit des délivrables de support aux tests (guides, modèles de documents, check listes ...), permet de définir une stratégie de test et définit qui doit faire quoi et comment. Son utilisation permettra d'éviter des situations d'urgence et de mieux déterminer le temps nécessaire aux tests. Cette démarche est une instanciation du modèle de processus de tests vu dans la deuxième partie Modélisation d'un processus de tests, pour être mis en oeuvre sur un petit projet interne en cours de développement.

La démarche est donc une proposition de méthodologie de mise en place des tests logiciels au CITI. Etant donnée la nature très diverse des projets de développement informatique réalisés au CITI, il est probable que tous les éléments de la démarche ne sont pas applicables tels quels sur tous les projets. Il faut adapter la démarche en fonction du type de projet (projet pour client externe/projet interne de recherche/taille du projet/complexité du fonctionnel ...). L'adaptation est à faire par le chef de projet, lors de la mise en place de l'organisation du projet.

J'ai présenté la démarche de la façon dont est présentée RUP, car elle décrit bien les activités à effectuer (avec des diagrammes d'activités d'UML), les rôles, les tâches et les responsabilités par personne affectées dans le projet. La démarche est donc présentée d'une façon telle que les personnes du projet savent ce qu'ils doivent faire, pourquoi ils doivent le faire et quand ils doivent le faire. De plus, j'ai créé les modèles de document, les check listes et un guide de création des cas de test allant avec la démarche (Voir les annexes, page 91).

7.2 Le cycle de vie de la démarche

La démarche proposée suit une approche processus (objectif du processus, éléments en entrée et en sortie, activités) itérative (Voir Figure 7.1 - Cycle de vie de la démarche) et permet donc un raffinement successif. Le cycle de vie des tests doit commencer en même temps que le développement du logiciel.

Chacune des itérations va être composée des phases suivantes :

- Planification des tests,
- Conception des tests,
- Exécution des tests,
- Evaluation des tests.

Cette approche par itération facilite les tests de non-régression. La plupart des tests de l'itération i-1 sont utilisés comme tests de non-régression à l'itération i. Puis les tests de l'itération i-1 et i sont utilisés comme tests de non-régression de l'itération i+1.

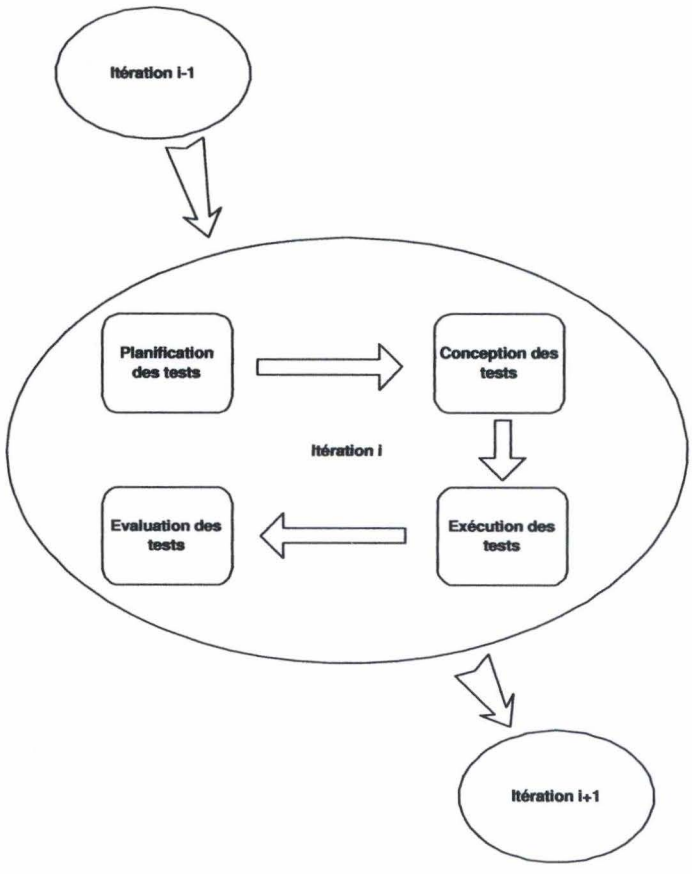


FIG. 7.1 – Cycle de vie de la démarche

Le cycle de vie de la démarche ne présente pas de phase d'implémentation des tests car le CITI n'a pas d'outil automatique dédié pour les tests. L'approche itérative permet de pratiquer les tests de non régression plus facilement. En effet, la grosse majorité des tests systèmes de l'itération précédente constitue les tests de non-régression de l'itération suivante. De plus, cette approche est bien adaptée au CITI qui pratique un développement logiciel en produisant des versions enrichies.

7.3 Les types de test

On pratique différents tests tout au long du développement du logiciel. Les étapes progressent du test du plus petit élément (un composant via le test unitaire), jusqu'au test du système complet intégré. (Voir Figure 7.2 - Etapes de test). Lors de chaque étape de test, on peut pratiquer plusieurs types de tests.

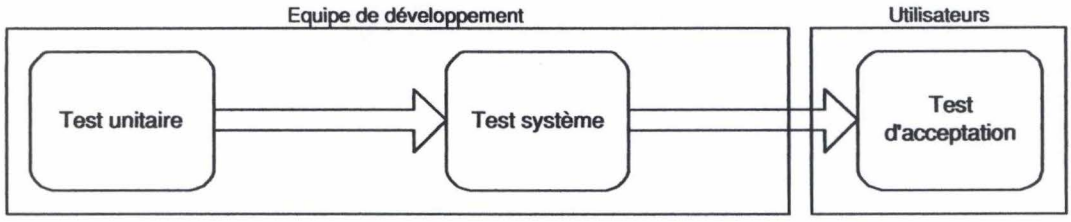


FIG. 7.2 – Etapes de test

Les différents projets peuvent choisir de faire tous les types qu'ils veulent sous les tests système, par exemple le choix de faire des tests de performance à cet endroit. L'étape des tests d'intégration ne fait pas non plus partie des tests prévus. Cela peut s'expliquer par le fait que l'on m'a demandé une version très légère de la démarche pour ne pas produire un effet de découragement, ni un effet de rejet à long terme, des développeurs en les contraignants à produire et à créer beaucoup de documents, et à effectuer beaucoup trop de tâches. Vu que certains projets sont développés que par quelques personnes, il a fallu aussi limiter le nombre de rôles possibles. Grâce à cette solution, chaque projet pourra pratiquer les tests nécessaires par rapport à son type.

7.3.1 L'étape de test unitaire

Le test unitaire se focalise sur la vérification du plus petit élément du logiciel. Plusieurs types de vérifications sont possibles comme les tests dits de la « boîte blanche » qui se basent sur la logique du programme et de la « boîte noire » qui se basent sur les spécifications fonctionnelles. Les règles et le suivi des tests unitaires sont définis dans le plan de test (Voir l'annexe B Modèle de document, page 107). Le test unitaire est effectué sur le composant par celui qui l'a créé : « On est responsable de ce que l'on produit ».

7.3.2 L'étape de test système

Les tests système permettent de tester un système d'éléments matériels et logiciels intégrés afin de prouver que le système répond aux exigences spécifiées. Lors de cette étape, on effectue des tests fonctionnels avec en parallèle des tests de performance, des tests de non-régression et des tests d'interface homme-machine.

Le test fonctionnel a pour objectif de vérifier la conformité du logiciel à ses spécifications. Les principaux objectifs des tests fonctionnels sont la conformité fonctionnelle sur base de spécifications, l'ergonomie, la sécurité et la détection des éventuelles carences de spécifications. Le test fonctionnel ignore la structure mais se focalise sur la fonctionnalité. Il est basé sur les exigences fonctionnelles du système. On teste une par une les fonctionnalités que le système devrait avoir selon la documentation des exigences fonctionnelles.

Les tests de performance ont pour objectif de vérifier la conformité du logiciel aux exigences non fonctionnelles comme, par exemple, le temps de réponse. Ils impliquent aussi bien le matériel que le logiciel. Différents tests de performance peuvent être réalisés comme les tests de surcharge, les tests de volume, les tests de séquençage et les tests de reprise, par exemple.

Les tests de non-régression vérifient en rejouant les tests déjà réalisés que le logiciel n'a pas été dégradé après une modification, c'est-à-dire l'ajout ou l'adaptation d'une fonction ou la correction d'un défaut.

Les tests IHM ont pour but de vérifier que la charte graphique et les règles ergonomiques ont été respectées tout au long du développement.

7.3.3 L'étape de test d'acceptation

Les tests d'acceptation ont pour objectif de vérifier que le logiciel est prêt, du point de vue des fonctionnalités, et qu'il peut être utilisé par les utilisateurs. Ces tests permettent aux utilisateurs de déterminer si le système est en accord avec leurs besoins et leurs attentes. C'est la dernière étape avant l'acceptation du système et sa mise en oeuvre opérationnelle. On teste le système dans les conditions définies par le futur utilisateur, plutôt que par le développeur. Les tests d'acceptation révèlent souvent des omissions ou des erreurs dans la définition de besoins. Il est possible que les besoins énoncés au début du projet ne reflètent pas les fonctionnalités réelles ou les vraies performances requises par l'utilisateur. Ces tests concernent des vérifications des fonctionnalités, de l'IHM et des performances.

7.4 Présentation des rôles

Le rôle définit la mission et les responsabilités d'une personne dans le projet. Un rôle peut être rempli par un ou plusieurs individus. Un individu peut lui-même remplir plusieurs rôles.

A chaque rôle est associé un ensemble d'activités et la responsabilité de certains documents.

Rôle	Description
Responsable des tests	Le responsable des tests est responsable du planning des tests et de générer le plan de test à chaque nouvelle itération de test. Il est également responsable de la stratégie de test et de l'allocation des ressources pour les tests. Ce rôle peut être assigné au chef de projet, au chef de projet de développement ou au responsable technique selon l'organisation du projet. Quelle que soit la personne à laquelle elle incombe, cette responsabilité doit être clairement définie.
Développeur	Il est responsable du développement des composants et des tests de ceux-ci. Il est responsable de la pratique des tests unitaires sur ses composants et de la correction des défauts détectés lors de tous les tests, et d'intégrer les composants au système pour les tests système.
Relecteur	Il est responsable d'assurer la qualité du code source et de mener la revue du code source. Il est aussi responsable de rédiger un rapport sur son activité de revue. Ce rôle est optionnel et dépend du temps et de l'effectif disponible. On peut mettre en oeuvre l'échange du code entre les développeurs.
Testeur	Il est responsable de l'exécution des tests système sur le système intégré. Généralement, il doit être indépendant de l'équipe de développement du projet. Dans de petites équipes de développement par manque/limite de ressources, les développeurs remplissent souvent le rôle de testeur.
Utilisateur	L'utilisateur est la personne qui va utiliser le système développé lorsque celui-ci sera livré. Il est responsable de la réalisation des tests d'acceptation.

TAB. 7.1 – Les rôles de la démarche

7.5 La présentation des délivrables

Le tableau 7.2 présente l'ensemble des délivrables utilisés lors de la mise en oeuvre de cette démarche. Il indique si la démarche propose un modèle de document, un guide ou une check-liste pour aider à l'élaboration de chaque délivrable.

Pour les modèles de document/« templates » des délivrables voir l'annexe B Modèle de document, page 107. Pour les check listes voir l'annexe C Check listes, page 117. Pour les guides voir l'annexe D Guide, page 119.

Délivrable	Définition	Modèle	Guide	Check liste
Cas d'utilisation	Ce document définit un ensemble d'instances de cas d'utilisation. Un cas d'utilisation est une séquence d'actions que le système accomplit et qui donne un résultat observable aux acteurs.	-	-	-
Cas de test	Ce document décrit les conditions particulières de réalisation d'un test spécifique. Il présente les données d'entrée à appliquer à l'application testée, les conditions d'exécution, et les résultats attendus suite à l'application de ces données lors de la réalisation du test.	X	X	X
Cas de test de performance	Ce document est utilisé pour réaliser les tests de performance. Il présente les contraintes non fonctionnelles à tester, que le système est censé offrir à l'utilisateur.	X	X	X
Cas de test fonctionnel	Ce document est utilisé pour réaliser les tests fonctionnels. Il présente toutes les fonctionnalités à tester, que le système est censé offrir à l'utilisateur.	X	X	X
Demande d'évolution	Les demandes d'évolution sont utilisées pour documenter et tracer les défauts et les demandes de modification. Elles fournissent un enregistrement des décisions et assurent une bonne compréhension des impacts dus au changement.	X	-	-
Document de suivi des tests	Ce document présente l'avancement des tests par rapport au plan de test.	-	-	-
Documents d'architecture	Ces documents fournissent une vue complète de l'architecture du système.	-	-	-
Fiche de livraison	Ce document décrit la version de l'application livrée dans le cadre du développement du logiciel. Y sont décrits la procédure d'installation, l'ensemble des nouvelles fonctionnalités présentes dans la version livrée, ce qui doit être testé en priorité, ce qui reste à réaliser et qui n'est pas à tester, la procédure de saisie des défauts par l'utilisateur.	X	-	-
Plan de test	Ce document contient les informations sur l'objectif et les buts de l'activité de tests sur le projet. De plus, le plan de test peut définir les stratégies pour exécuter les tests, ainsi que les ressources nécessaires aux tests. Il fixe aussi les règles que le développeur doit suivre pour les tests unitaires.	X	-	X
Planning du projet	Ce document représente le séquencement dans le temps des activités et des tâches, les ressources assignées et les dépendances entre les tâches du projet.	-	-	-
Rapport de relecture	Ce document présente l'ensemble des défauts du code qui a été revu. Il est transmis au programmeur pour la correction éventuelle du code.	-	-	-
Spécifications supplémentaires	Ce document présente les exigences systèmes que les cas d'utilisation ne représentent pas.	-	-	-

TAB. 7.2 – Les délivrables de la démarche

7.6 Les activités et les tâches de la démarche

7.6.1 Le diagramme d'activités de la démarche

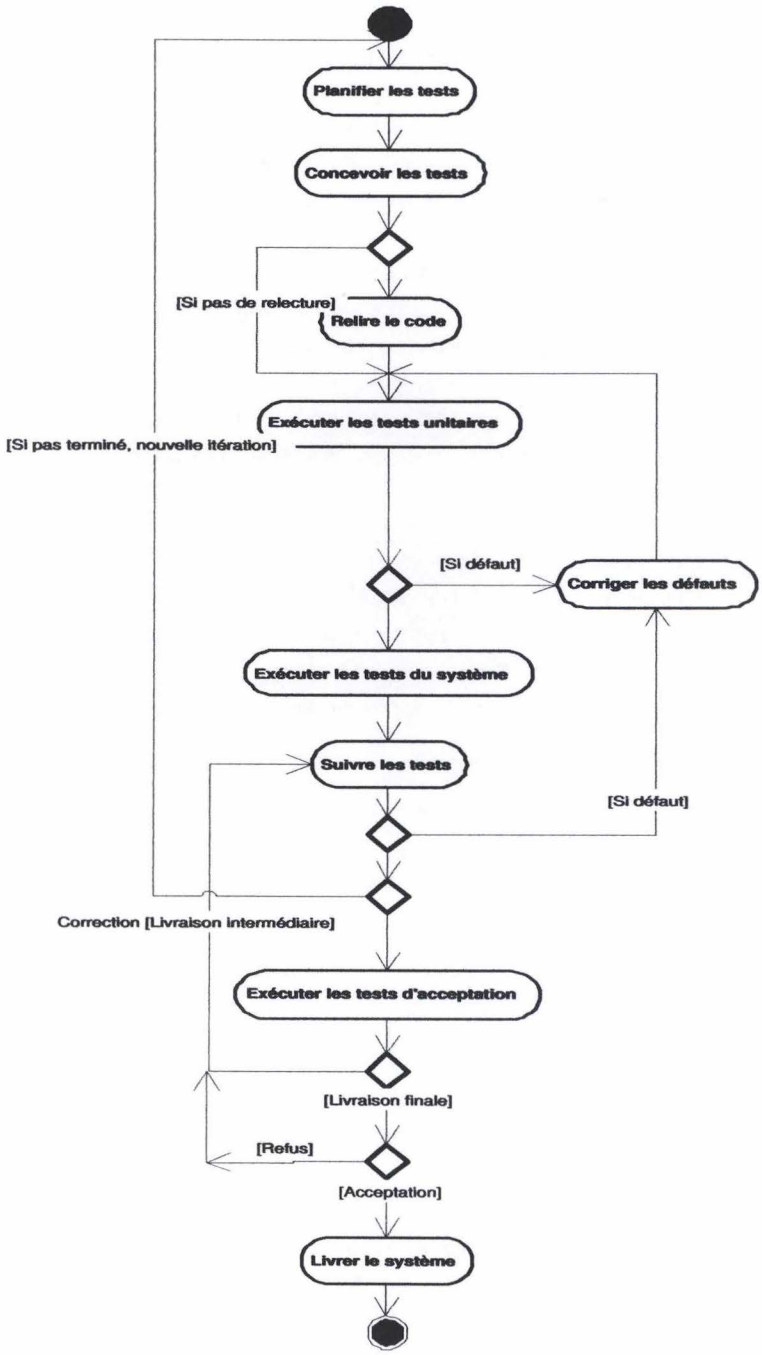


FIG. 7.3 – Le diagramme d'activités de la démarche

Chaque activité et lien entre les activités sont décrits dans les points suivants.

7.6.2 Planifier les tests

Présentation de l'activité

Le but de cette activité est, tout d'abord, de collecter l'information sur la planification des tests, et, ensuite, de créer un plan de test. (Voir Figure 7.4 - Diagramme d'activités de planification)

La personne responsable de cette activité est le responsable des tests.

Les éléments en entrée et en sortie sont :

Entrée	Sortie
Documents d'architecture	Plan de test
Planning de projet	-
Cas d'utilisation	-
Spécifications supplémentaires	-
Modèle de plan de test	-
Check liste de plan de test	-

TAB. 7.3 – Entrées-sorties de l'activité de planification des tests

Le support pour la rédaction du plan de test est une check liste. Le plan de test est créé à l'aide d'un modèle de document.

Les supports d'UML sont les diagrammes de cas d'utilisation et de séquence.

L'activité suivante est de concevoir les tests.

Les tâches liées à cette activité sont de :

- Identifier les exigences pour les tests, c'est-à-dire identifier ce qui doit être testé et indiquer la portée des tests ;
- Etablir une séquence acceptable de test ;
- Développer une stratégie de test, c'est-à-dire identifier et communiquer les techniques et les outils de test, identifier et communiquer la méthode d'évaluation pour déterminer la qualité du produit et l'exécution des tests. Il sera défini en particulier les règles de préparation et d'exécution des tests unitaires ;
- Identifier les ressources nécessaires pour les tests, c'est-à-dire les ressources humaines, matérielles, logicielles et les outils ;
- Créer un planning pour identifier et communiquer l'effort de test, le calendrier et les points clefs ;
- Et enfin, rédiger le plan de test à l'aide du modèle.

Diagramme de l'activité de planification

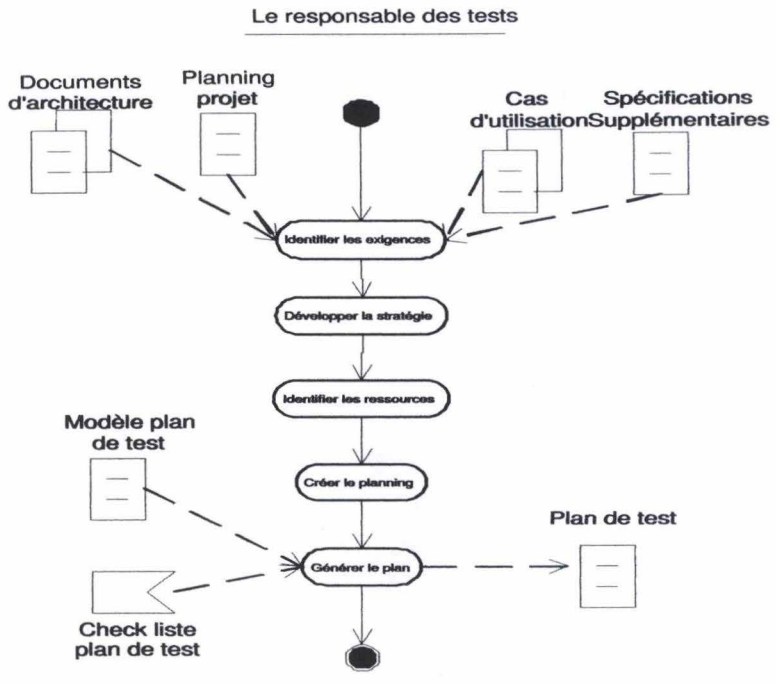


FIG. 7.4 – Le diagramme d'activités de planification

7.6.3 Concevoir les tests

Présentation de l'activité

Le but de cette activité est d'identifier et rédiger un ensemble de cas de test afin de vérifier le fonctionnement du système. (Voir Figure 7.5 - Diagramme d'activités de conception)

La personne responsable de cette activité est le testeur pour les tests système. Il peut s'agir du testeur et/ou de l'utilisateur pour les tests d'acceptation. L'utilisateur doit faire partie des employés impliqués dans la définition des exigences pour avoir une bonne connaissance du domaine.

Les éléments en entrée et en sortie sont :

Entrée	Sortie
Cas d'utilisation	Cas de test
Guide de création de cas de test	-
Plan de test	-
Spécifications supplémentaires	-
Documents d'architecture	-
Check liste de cas de test	-
Modèle de cas de test	-

TAB. 7.4 – Entrées-sorties de l'activité de conception des tests

Les supports pour la rédaction des cas de test sont une check liste, un modèle et un guide

de rédaction.

Les supports d'UML sont :

	Diagrammes UML
Conception des tests	Diagramme des cas d'utilisation pour la conception des cas de test. Diagramme de classe pour la structure hiérarchique des classes. Diagramme de séquence, pour les tests de performance, pour avoir une vue temporelle et des messages. Diagramme d'activité pour la synchronisation et les responsabilités. Diagramme de composants pour identifier les composants. Diagramme d'états pour avoir une vue sur les changements d'état.

TAB. 7.5 – Support d'UML pour la conception des tests

L'activité précédente est de planifier les tests et l'activité suivante est soit de relire le code, soit d'exécuter les tests unitaires.

Les tâches liées à cette activité sont de :

- Identifier et décrire les cas de test système (fonctionnels et de performance) à l'aide du modèle et de la check liste des cas de test.
- Identifier et décrire des cas de tests d'acceptation sur la base des cas de tests existants en favorisant les scénarii d'utilisation habituelle.

Diagramme de l'activité

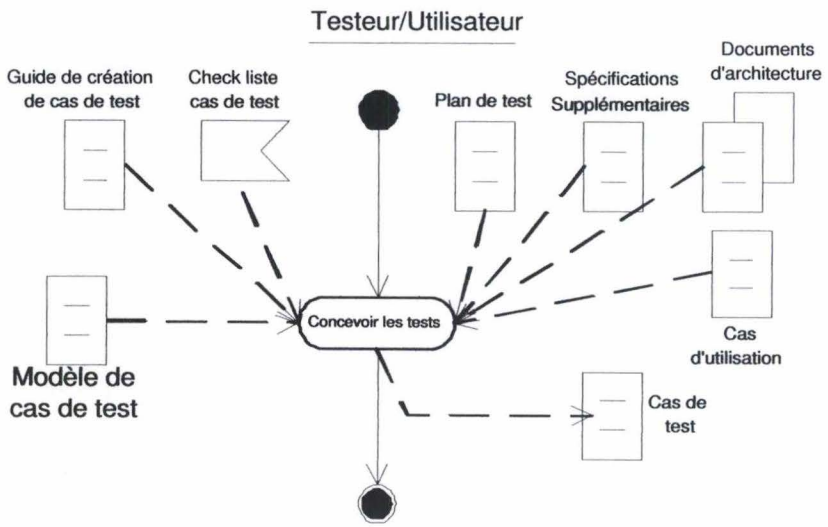


FIG. 7.5 – Le diagramme d'activités de conception

7.6.4 Relire le code

Présentation de l'activité

Le but de cette activité est la relecture du code source d'un « composant » produit par un développeur pour le vérifier avant de le tester. (Voir Figure 7.6 - Diagramme d'activités de relecture du code)

La personne responsable de cette activité est le relecteur. Si l'équipe n'est pas assez grande pour avoir un relecteur attitré, on peut pratiquer l'échange de code. Chaque développeur échange son code avec un autre qui le relit et lui fournit un rapport de relecture.

Les éléments en entrée et en sortie sont :

Entrée	Sortie
Code source du composant	Rapport de relecture

TAB. 7.6 – Entrées-sorties de l'activité de relecture du code

Cette activité peut être réalisée avant ou après l'exécution des tests unitaires.

Les tâches liées à cette activité sont de :

- Examiner le code du composant ;
- Générer un rapport de relecture ;
- Fournir le rapport au développeur du composant.

Diagramme de l'activité

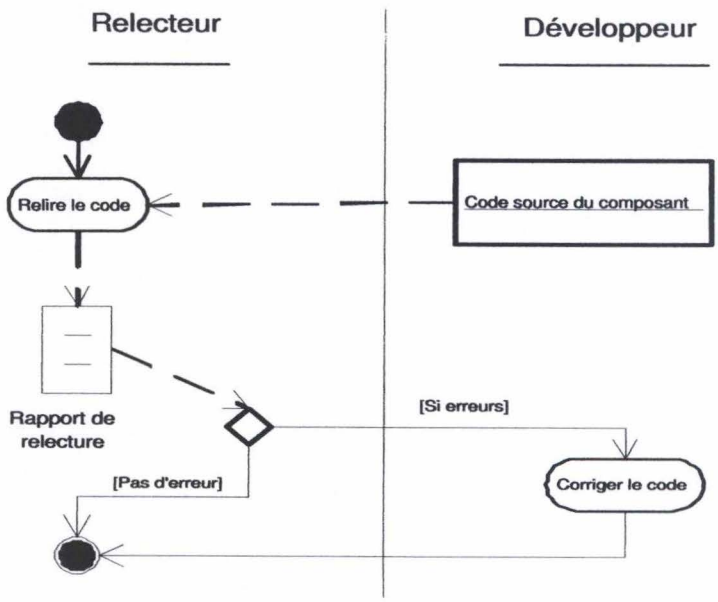


FIG. 7.6 – Le diagramme d'activités de relecture du code

7.6.5 Exécuter les tests unitaires

Présentation de l'activité

Le but de cette activité est d'exécuter les tests et de capturer les résultats, c'est-à-dire de vérifier la spécification et la structure d'une unité/composant. (Voir Figure 7.7 - Diagramme d'activités d'exécution des tests unitaires)

La personne responsable de cette activité est le développeur. Celui-ci est responsable de la préparation et de l'exécution des tests unitaires mais il doit les organiser et les réaliser dans le respect des règles et de la stratégie définies dans le plan de test.

Les supports d'UML sont :

	Diagrammes UML
Tests unitaires	Diagramme des cas d'utilisation pour la conception des tests. Diagramme de classe pour représenter les relations entre classes. Diagramme de collaboration pour montrer les interactions entre objets. Diagramme d'états pour avoir une vue sur les changements d'état.

TAB. 7.7 – Support d'UML pour les tests unitaires

Les éléments en entrée et en sortie sont :

Entrée	**Sortie**
Cas d'utilisation	Composant (modifié)
Plan de test	-
Spécifications supplémentaires	-
Documents d'architecture	-
Composant	-

TAB. 7.8 – Entrées-sorties de l'activité d'exécution des tests unitaires

L'activité suivante est d'exécuter les tests système.

Les tâches liées à cette activité sont de :

- Exécuter les tests unitaires ;
- Vérifier les résultats des tests ;
- Corriger les défauts.

Diagramme de l'activité d'exécution des tests unitaires

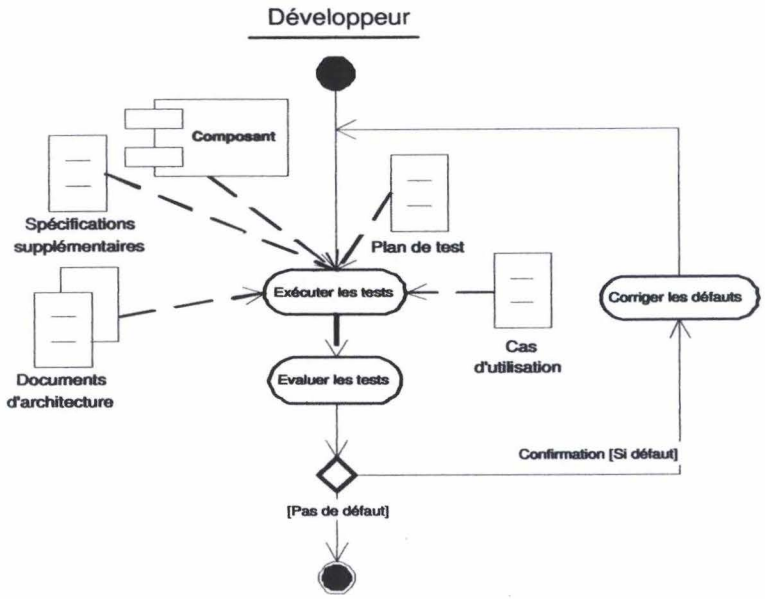


FIG. 7.7 – Le diagramme d'activités d'exécution des tests unitaires

7.6.6 Exécuter les tests système

Présentation de l'activité

Le but de cette activité est l'exécution et la capture des résultats des tests système. (Voir Figure 7.8 - Diagramme d'activités d'exécution des tests système)

Les personnes responsables de cette activité sont le testeur et le développeur.

Les éléments en entrée et en sortie sont :

Entrée	Sortie
Cas de test système	Demandes d'évolution
Système intégré	-

TAB. 7.9 – Entrées-sorties de l'activité d'exécution des tests système

Le support pour cette activité est la base de collecte de demandes d'évolution.

L'activité précédente est d'exécuter les tests unitaires et l'activité suivante est de suivre les tests.

Les tâches liées à cette activité sont de :

- Intégrer les composants avec l'actuel système pour en faire le système intégré ;
- Exécuter les tests sur le système intégré ;
- Saisir les demandes d'évolution.

Les outils disponibles sont des bases de collecte et de revue de demandes d'évolution :

- La base Notes « Rapports de Tests » ;
- L'application de gestion des anomalies Mantis.

Diagramme de l'activité d'exécution des tests système

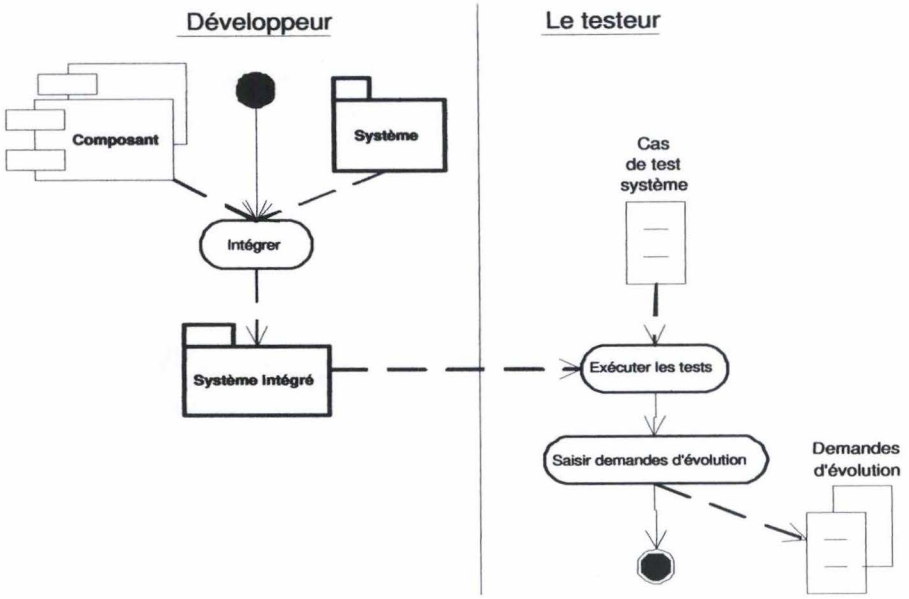


FIG. 7.8 – Le diagramme d'activités d'exécution des tests système

7.6.7 Suivre les tests

Présentation de l'activité

Le but de cette activité est d'évaluer les résultats des tests système et d'acceptation, de passer en revue les demandes d'évolution pour leur donner un responsable et une priorité et de rédiger un récapitulatif de l'avancement des tests. (Voir Figure 7.9 - Diagramme d'activités de suivi des tests)

La personne responsable de cette activité est le responsable des tests.

Les éléments en entrée et en sortie sont :

Entrée	Sortie
Demandes d'évolution	Document de suivi des tests
-	Demandes d'évolution (modifiée)

TAB. 7.10 – Entrées-sorties de l'activité de suivi des tests

Le support pour cette activité est la base de collecte de demandes d'évolution.

L'activité précédente est d'exécuter les tests système.

Les supports UML pour l'évaluation des tests sont le diagramme d'état de la demande d'évolution.

Les tâches liées à cette activité sont de :

- Revoir les demandes d'évolution ;
- Evaluer les demandes d'évolution en leur assignant un responsable, une priorité, un impact, une estimation de temps ;
- Modifier les demandes d'évolution ;
- Identifier l'avancement des tests par rapport au plan de test ;
- Produire le document récapitulatif de suivi des tests.

Diagramme de l'activité de suivi des tests

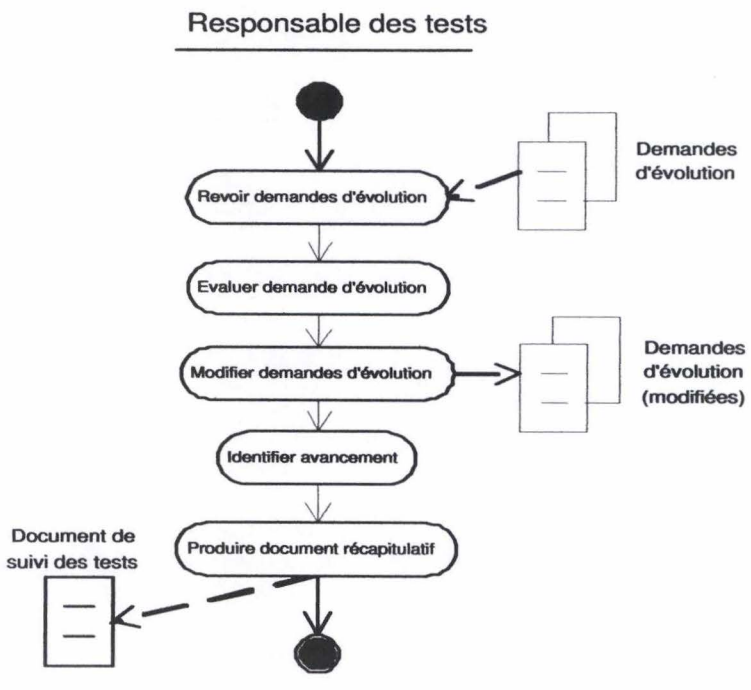


FIG. 7.9 – Le diagramme d'activités de suivi des tests

Diagramme d'états de la demande d'évolution dans la base Notes

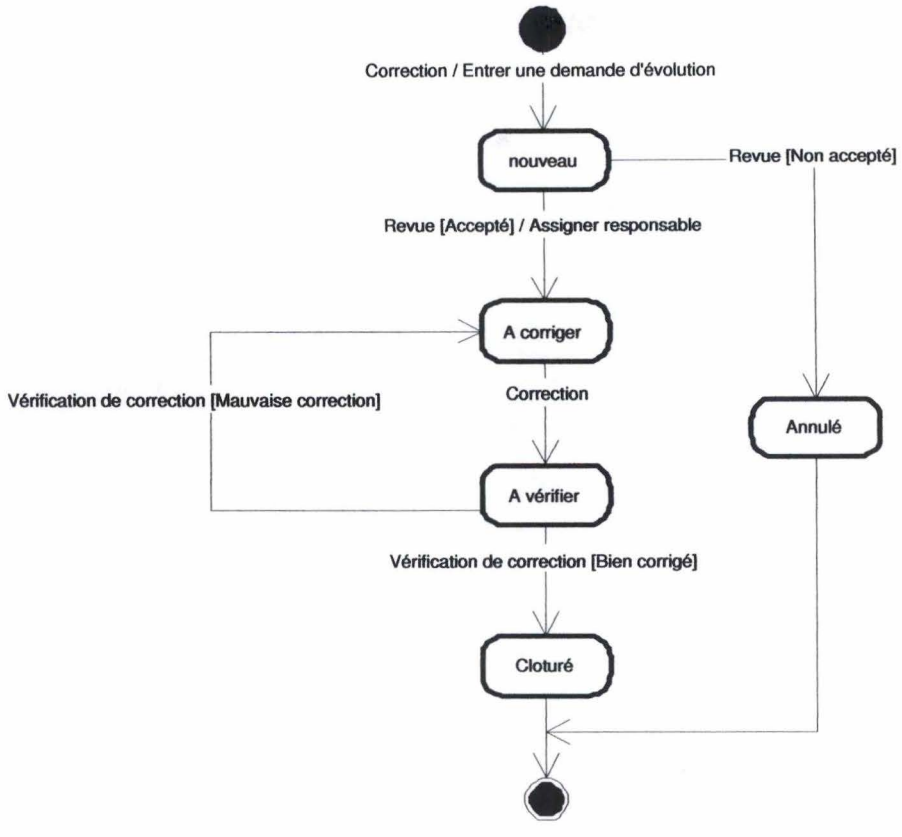


FIG. 7.10 – Le diagramme d'état demande d'évolution Base Notes

Lorsqu'une nouvelle demande d'évolution d'anomalie ou de correction est entrée, elle peut être acceptée est alors passez en revue. Sinon, elle est non acceptée et est, dans ce cas, annulée. Après être acceptée pour la revue, elle passe en correction. La correction est ensuite vérifiée pour voir si elle est bien corrigée. Si elle est bien corrigée alors elle est clôturée sinon elle passe à nouveau en correction pour cause de mauvaise correction.

Diagramme d'états de la demande d'évolution dans Mantis

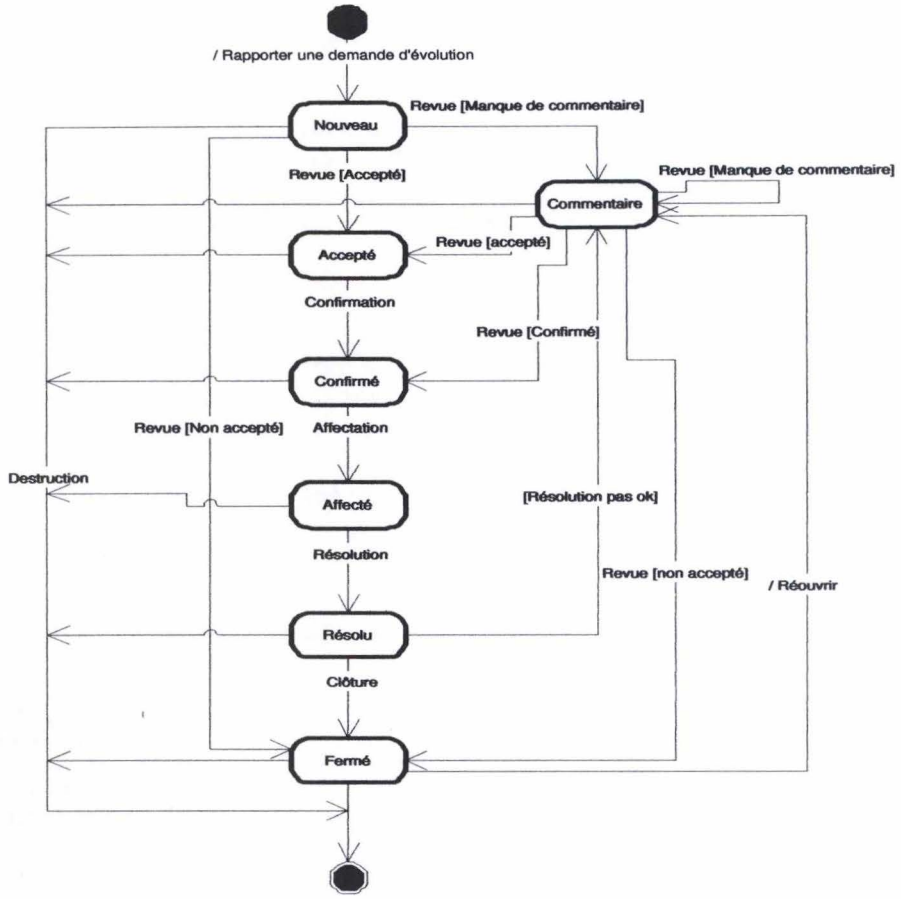


FIG. 7.11 – Le diagramme d'états de la demande d'évolution dans Mantis

Lorsqu'une nouvelle demande d'évolution pour cause d'anomalie ou de correction est rapportée dans le système Mantis, elle est revue pour voir la véracité du rapport. S'il y a un manque de commentaires, elle est renvoyée en attendant plus de commentaires de la part du rapporteur. Si elle est acceptée, elle est ensuite confirmée et affectée à un développeur pour résoudre le problème. Lorsqu'elle est résolue, la demande est fermée. A toutes les étapes, la demande peut être détruite si elle est jugée non critique ou qu'elle ne présente aucune anomalie.

Diagramme d'états de la demande d'évolution : le flux idéal

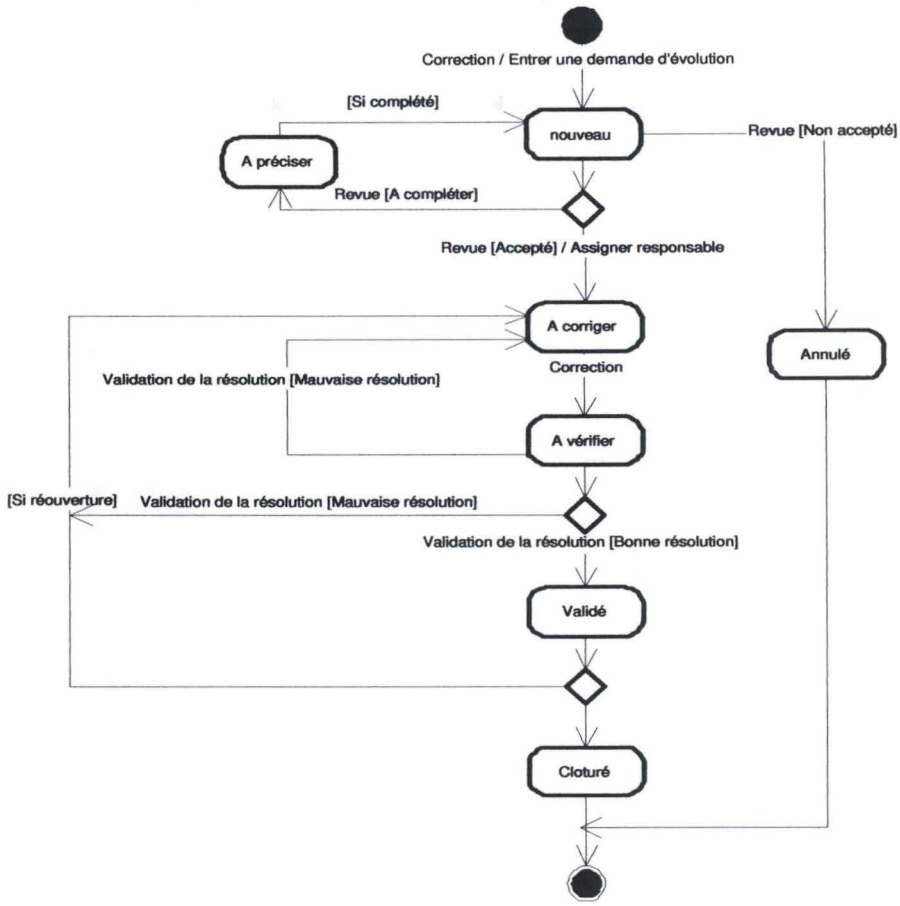


FIG. 7.12 – Le diagramme d'états de la demande d'évolution du flux idéal

Lorsqu'une nouvelle demande d'évolution pour cause d'anomalie ou correction est revue pour voir s'il n'y a pas d'élément à préciser. Si la demande est refusée alors elle est annulée. Si, par contre, elle est acceptée, elle est assignée à un responsable pour la correction. Après, la correction de la demande est vérifiée et est à nouveau renvoyée en correction si celle-ci s'avère mauvaise. Dans l'autre cas, elle est validée et ensuite clôturée. La demande peut être réouverte si l'anomalie se représente.

7.6.8 Corriger les défauts

Présentation de l'activité

Le but de cette activité est de corriger les défauts signalés dans les demandes d'évolution. (Voir Figure 7.13 - Diagramme d'activités de correction des défauts)

La personne responsable de cette activité est le développeur.

Les éléments en entrée et en sortie sont :

Entrée	Sortie
Demandes d'évolution	Demandes d'évolution (modifiées)
Composant	Composant (corrigé)

TAB. 7.11 – Entrées-sorties de l'activité de correction des défauts

Le support pour cette activité est la base de collecte des demandes d'évolution.

L'activité précédente est de suivre les tests.

Les supports d'UML sont les diagrammes de classe pour les problèmes concernent les interactions entre classe, les diagrammes de collaboration concernent les interactions entre objets, les cas d'utilisation concernent les fonctionnalités, et l'ensemble des modèles de conception et d'implémentation.

Les tâches liées à cette activité sont de :

- Revoir les demandes d'évolution ;
- Apporter les corrections au code ;
- Refaire les tests unitaires ;
- Modifier les demandes d'évolution.

Diagramme de l'activité de la correction des défauts

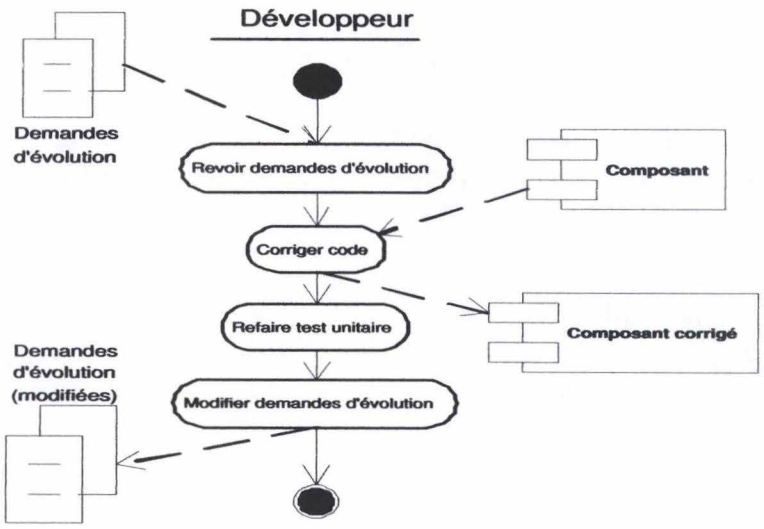


FIG. 7.13 – Le diagramme d'activités de correction des défauts

7.6.9 Exécuter les tests d'acceptation

Présentation de l'activité

Le but de cette activité est de mener les tests d'acceptation sur base de cas de test préparés au préalable lors de l'activité de conception des tests. (Voir Figure 7.14 - Diagramme d'activité

d'exécution des tests d'acceptation)

La personne responsable de cette activité est l'utilisateur.

Les éléments en entrée et en sortie sont :

Entrée	Sortie
Cas de test d'acceptation	Demandes d'évolution
Système livré	-

TAB. 7.12 – Entrées-sorties de l'activité d'exécution des tests d'acceptation

L'activité précédente est de suivre les tests et l'activité suivante est soit de suivre les tests, soit de livrer le système.

Les tâches liées à cette activité sont de :

- Exécuter les tests d'acceptation ;
- Saisir les demandes d'évolution.

Diagramme de l'activité d'exécution des tests d'acceptation

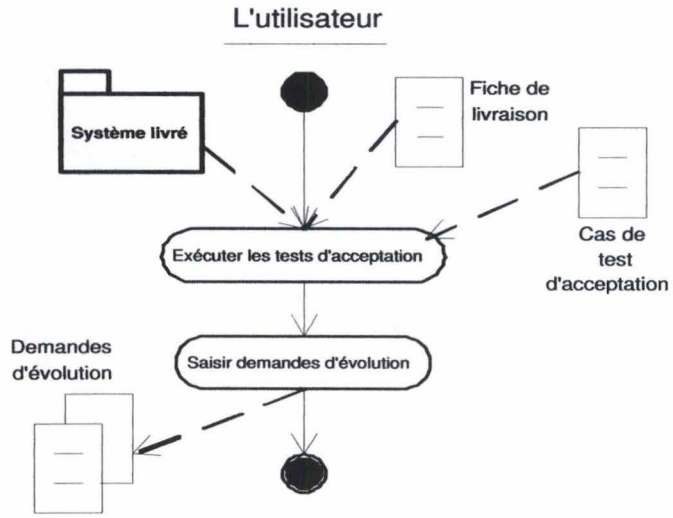


FIG. 7.14 – Le diagramme d'activités d'exécution des tests d'acceptation

7.7 Le diagramme d'activité de la démarche

Ce diagramme reprend tous les diagrammes d'activités pour donner une vue générale des activités liées à la démarche.

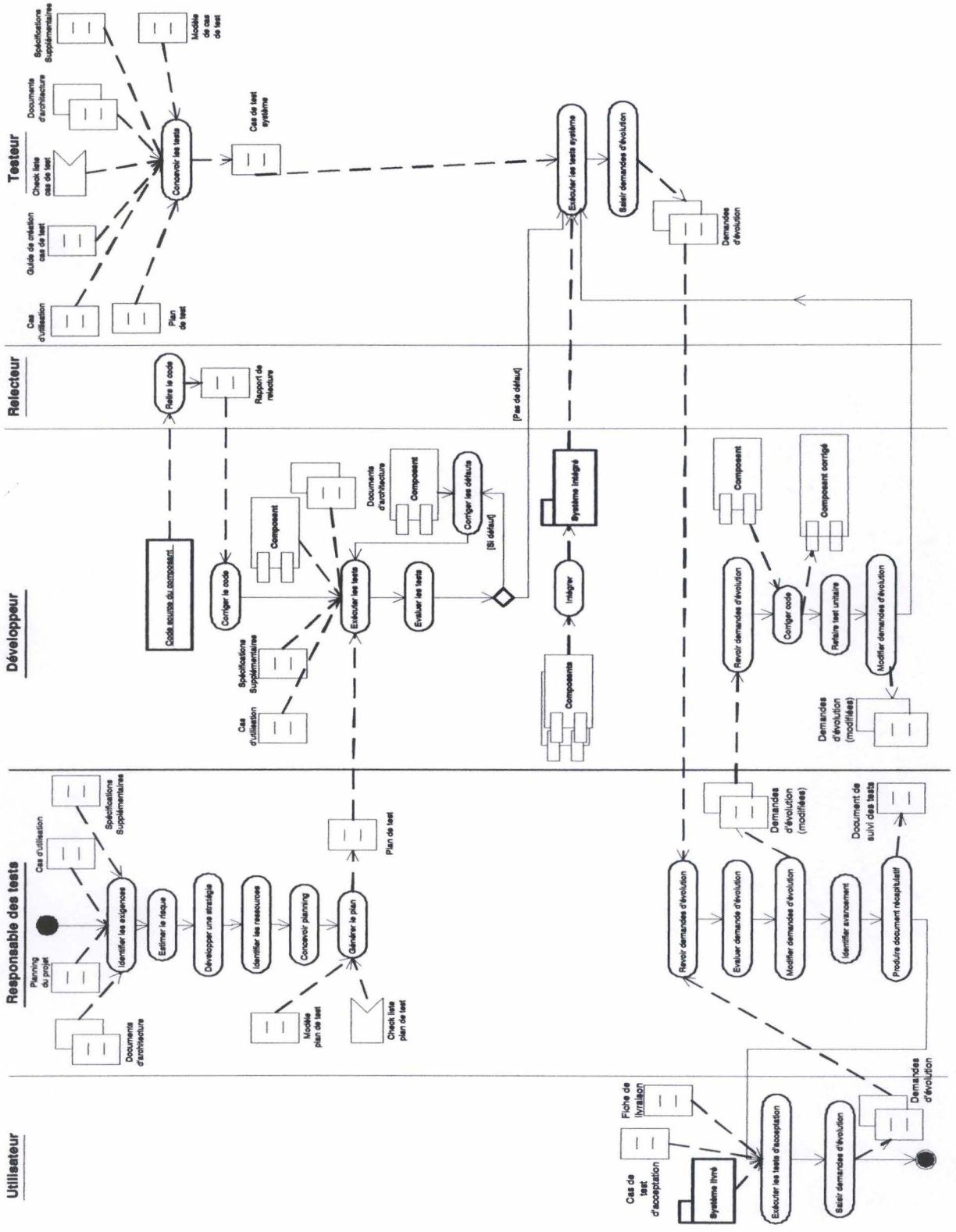


FIG. 7.15 – Le diagramme d'activités de la démarche

Conclusion

Une démarche de tests embryonnaire a été faite avec une classification des tests, un cycle de vie, les documents à remplir et à créer, les activités nécessaires à la réussite des tests, les rôles et les responsabilités que chaque personne participant au projet doit effectuer, les modèles de documents et guide de cas de test. Embryonnaire car elle doit faire prendre conscience aux participants de l'utilité de gérer efficacement les tests. Cette démarche constitue donc un guide de démarrage du processus de tests.

La démarche n'est pas complète. En effet, il manque une partie d'implémentation des tests par des outils automatiques qui peuvent aider les testeurs dans leurs tâches surtout dans les tests de non-régression, tests qui sont répétitifs. La partie sur UML n'est que dans sa phase d'idée. En effet, il faudrait effectuer une enquête sur des testeurs qui devraient utiliser la modélisation UML pour les aider dans les tests. Avec ce moyen, on pourra déterminer quel diagramme est effectivement utile dans ce type de tests, ou pourquoi ne pas créer leur propre diagramme.

La démarche de tests logiciels, telle que proposée au CITI, va passer une nouvelle phase d'expérimentation. Cette phase a pour but de mettre en oeuvre la démarche de tests lors d'un projet de développement interne au CITI. Cette démarche a pour objectif de montrer l'importance de planifier et de formaliser le processus de tests. L'équipe SPINOV espère qu'après l'expérimentation et les rapports fournis, elle va permettre l'éclosion d'une démarche de tests logiciels adaptables, selon les différents développements du CITI, à partir de la démarche « embryonnaire » que j'ai développée. En effet, la démarche est la moins élaborée possible, en termes de documents, de types de tests, pour ne pas avoir un rejet de la démarche. La démarche va pouvoir, si elle est acceptée au sein du CITI, soulager l'utilisateur qui était le principal testeur, en majorité, des logiciels développés. Elle pourra aussi faire prendre conscience aux participants qu'il est possible de gérer les tests à l'avance et ne plus se retrouver dans des situations d'urgence.

Lorsque le processus de tests est bien défini, il permet de bien organiser les tests dans le projet. Grâce à cela, l'équipe de tests sait ce qu'elle doit faire, pour quand et pour qui, comment elle doit le faire et dans quel but. Jamais personne ne produira un document ou n'effectuera une tâche qui ne sert à rien. Il permet aussi un contrôle sur les tâches, faites ou non, ainsi qu'une traçabilité de la documentation. Il permet aussi d'avoir un produit logiciel de plus grande qualité, ou de prendre des décisions au niveau de la qualité à temps, pour éviter que le client ne soit confronté à un logiciel inutilisable.

Un processus de tests doit être adapté au type de projet. Plus le projet est critique ou important, plus le processus doit être complet. Les projets de petite taille et non critiques peuvent définir une dizaine d'activités et une dizaine de documents à créer ou à remplir. Par contre, un projet beaucoup plus grand ou critique, avec une grande équipe de développeurs, devrait avoir un processus de tests plus élaboré.

L'aspect le plus important qui ressort de ce mémoire est de ne pas négliger le contexte du projet. Que ce soit pour la méthodologie ou les tests logiciels, il faut pour chaque projet analyser au préalable les facteurs critiques tels que la taille du projet, sa criticité, son dynamisme, l'expérience du personnel mis en place et la culture d'entreprise, sans oublier les éléments comme les usagers, les biens livrables, l'équipe de développement et l'environnement de développement.

Bibliographie

[Ama03] K. Amamra. *The Capability Maturity Model Integration.* http ://www.univ-angers.fr/docs/etudquassi/CMMI.pdf, DESS QUASSI 2002-2003.

[Anc04] F. Anceau. *Techniques de test et validation du logiciel.* http ://lmil7.cnam.fr/ anceau/Documents/test1.pdf, date d'accès avril 2004.

[ASRW02] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. *Agile Software Development Methods : Review and Analysis.* VTT Publications 478, Espoo, 2002.

[Ber04] A. Bertolino. *Guide to the Software Engineering Body of Knowledge.* http ://www.swebok.org, IEEE - Trail Version 1.00, date d'accès février 2004.

[Cae03] J. Caelen. *Rationaliser la conception participative.* http ://www.geod.imag.fr/jcaelen/transparents/, date d'accès octobre 2003.

[CMM94] CMM. *Software Process Maturity Questionnaire, Capability Maturity Model.* version 1.1, avril 1994.

[CMM02] CMMi. *CMMi for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing.* mars 2002.

[CRI04] CRIM. *Test logiciel.* http ://www.crim.ca/index.epl/selec/ctl/services/tests-logiciel.htm, date d'accès avril 2004.

[Dan04] S. Dangbo. *Organiser des tests dans un projet.* http ://www.igm.univ-mlv.fr/ dr/XPOSE/TesTs/SiteWeb/typestets.htm, date d'accès avril 2004.

[Dic04] Dictionnaire. *Définition.* http ://www.ledico.net/definitions/, date d'accès avril 2004.

[Enc04] Encyclopédie. *Définition.* http ://encyclopedie.journalunet.com/definition, date d'accès avril 2004.

[eodsd87] Collection Mise en oeuvre de système d'information. *Guide de gestion de projet : Gérer le développement par produits.* Groupe DMR Inc., troisième edition, 1987.

[eP04] eXtreme Programming. *eXtreme Programming.* http ://www.extremeprogramming.org, date d'accès février 2004.

[FL00a] R. Fannader and H. Leroux. *UML : Principes de mise en oeuvre.* Dunod, 2000.

[FL00b] R. Fannader and H. Leroux. *UML : Principes de modélisation.* Dunod, 2000.

[fra98] Norme française. *Evaluation de processus du logiciel, partie 5 : Un modèle d'évaluation et guide des indicateurs.* AFNOR, 1998.

[Gal04] F. Di Gallo. *Cours de génie logiciel - Cycle probatoire.* http ://fdigallo.online.fr/cours/génie-logiciel.pdf, date d'accès avril 2004.

[GG98] T. Gilb and D. Graham. *Software Inspection.* Addison-Wesley, Edition 1998.

[HA04] N. Habra and S. Alexandre. *Ingénierie du logiciel, matière approfondie.* cours d'ingénierie du logiciel matière approfondie, 2004.

[HR04] N. Habra and A. Renault. *Modèle OWPL : Evaluation et amélioration des pratiques logicielles dans les PME wallonnes.* http ://www.info.fundp.ac.be/ software-quality/fr/owpl/doc/OWPL-Modele.pdf, version 1.2.2 b de 2001, date d'accès mai 2004.

[ISO02] ISO/CEI. *Ingénierie du logiciel-Evaluation du produit logiciel-Exigences qualité pour les progiciels et instructions d'essai.* ISO/WD 12119 5, ISO, 2002.

[Kon04] Projet Kontikit. *Tests logiciels.* http ://dess-gla.infop6.fussieu.fr/ kontikit/content/techndocs/GL, date d'accès avril 2004.

[Lar00] Larousse. *Dictionnaire Larousse de poche.* Larousse, Edition 2000.

[Mar03] J. Marant. *Document de synthèse sur XP.* http ://www.idealx.org/doc/xp-synthese.fr.html, date d'accès octobre 2003.

[Pet04] M. Petit. *Informatique de base - Techniques de tests.* http ://www.info.fundp.ac.be/ mpe/, date d'accès avril 2004.

[Pfl01] S. L. Pfleeger. *Software Engineering : Theory and Pratice.* Prentice Hall, seconde edition, 2001.

[Rat01] Rational. *Rational Solution for Windows.* sous forme de CD-Rom, version mai 2001.

[Som04] S. S. Somé. *Intégration du système.* http ://www.site.uottawa.ca/ ssome/cours/SEG4511/, date d'accès avril 2004.

[Tol04] M. Tollenaere. *Chapitre 7 : démarche de conception, conduite de projet SI.* http ://gilco.inpg.fr/tollenaere/msi/diaporamas/SI_intro.ppt, date d'accès août 2004.

[Vic04] J.-P. Vickoff. *Méthode de développement logiciel RAD.* http ://our-world.compuserve.com/homepages/vickoff/phasprin.htm & http ://ma-page.noos.fr/rad/radmethod.pdf, date d'accès avril 2004.

[Wil04] C. E. Williams. *Software Testing and the UML.* http ://www.chillarege.com/fastabstracts/issre99/ 99119.pdf, date d'accès avril 2004.

Quatrième partie

Annexes

Annexe A

Questionnaires et évaluation

A.1 Questionnaire SPICE sur les tests logiciels au CITI

Résumé

Ce document est un questionnaire basé sur le modèle SPICE pour évaluer la façon dont les tests logiciels sont pratiqués au CITI.

Processus de vérification

Développer une stratégie de vérification

Il y a-t-il une stratégie de vérification développée ?

Les critères de vérification sont-ils spécifiés pour tous les produits de travail concernés ?

Mener les vérifications

Est-ce que les produits de travail identifiés conformément à la stratégie ont-ils été vérifiés ?

Déterminer les actions sur les résultats de la vérification

Est-ce que les problèmes identifiés lors de la vérification ont-ils été analysés ?

Est-ce que les actions pour permettre de les résoudre ont été déterminées ?

Suivre les actions sur les résultats de la vérification

Les actions sur les résultats ont-elles été suivies ?

Est-ce que les résultats ont été rendus disponibles aux clients ?

Est-ce que les résultats ont été rendus disponibles aux organisations concernées ?

Processus d'intégration du logiciel

Développer une stratégie d'intégration du logiciel

Y a-t-il une stratégie prévue pour l'intégration des modules du logiciel ?

Est-elle cohérente avec les livrables ?

Y-a-t-il identification de paquetage du logiciel ?

Y-a-t-il une séquence/ordre défini pour les vérifier ?

Développer une stratégie de test de non-régression des composants de logiciel intégrés

Y-a-t-il une stratégie de test de non-régression prévue en cas de changement d'un composant dans un module ?

Développer les tests des composants de logiciel intégrés

Y-a-t-il une description des tests exécutés pour chaque élément de logiciel intégré ?

Est-ce que les exigences du logiciel à contrôler sont précisées ?

Est-ce que les critères de vérification ont été précisés ?

Vérifier les composants de logiciel intégrés

Y-a-t-il des critères d'acceptation pour chaque élément ?

Est-ce que les composants de logiciel intégrés ont été intégrés en utilisant ces critères d'acceptation ?

Les résultats ont-ils été documentés ?

Intégrer des composants de logiciels

Est-ce que la cohérence entre les exigences et la conception du logiciel a été assurée ?

Vérifier la non-régression des composants du logiciel intégrés

Lors d'un changement d'un module de logiciel, y-a-t-il exécution des tests de non-régression tel que définis dans la stratégie de test de non-régression ?

Processus d'essai du logiciel

Développer une stratégie de test du logiciel intégré, incluant une stratégie de non-régression

Y-a-t-il une stratégie de test de non-régression prévue en cas de changement d'un module dans le logiciel intégré ?

Développer les tests pour le logiciel intégré

Y-a-t-il description des tests à exécuter pour le produit logiciel complet ?

Les exigences logicielles à contrôler ont-elles été indiquées ?

Idem pour les entrées et les critères d'acceptation ?

Est-ce que l'ensemble des tests démontre la conformité aux exigences du logiciel ?

Vérifier le logiciel intégré

Est-ce que le logiciel intégré a été vérifié en utilisant les critères d'acceptation ?

Est-ce que les résultats ont été documentés ?

Vérifier la non-régression du logiciel intégré

Lors d'un changement d'un module du logiciel, y-a-t-il exécution des tests de non-régression tel que définis dans la stratégie de test de non-régression ?

Processus d'intégration et d'essai du système

Développer une stratégie d'intégration et de test du système

Une stratégie d'intégration du système a-t-elle été effectuée ?

Une stratégie de test du système a-t-elle été effectuée ?

Construire les paquetages de système

Des paquetages ont-ils été identifiés ?

Un ordre/séquence pour les tests a-il été établi ?

Développer les tests pour les paquetages du système

Les tests à réaliser pour chaque paquetage ont-ils été décris ?

Les exigences à contrôler ont-elles été indiquées ?

Les données d'entrée ont-elles été indiquées ?

Les composants de système nécessaires pour réaliser les tests ont-ils été indiqués ?

Les critères d'acceptation ont-ils été indiqués ?

Vérifier chaque paquetage de système

Chaque paquetage de système satisfait-il les exigences ?

Les résultats ont-ils été documentés ?

Développer les tests pour le système

Les tests pour le système intégré à réaliser ont-ils été décris ?

Les exigences à contrôler ont-elles été indiquées ?

Les données d'entrée ont-elles été indiquées ?

Les critères d'acceptation ont-ils été indiqués ?

Vérifier le système intégré

Des vérifications du logiciel intégré ont-elles été pratiquées pour assurer la satisfaction des exigences du système ?

Les résultats ont-ils été documentés ?

Vérifier la non-régression des paquetages du système ou le système intégré

Lors d'un changement d'un composant du système, y-a-t-il exécution des tests de non-régression tel que définis dans la stratégie de test de non-régression ?

Questions sur les attributs

Process Performance (PP)

The extent to which the process uses a set of activities which are initiated and performed using identifiable input work products which are adequate to meet the defined process outcomes. Note : This attribute addresses whether the process is complete i.e. whether a set of activities and associated tasks are performed that produce work products that achieve the process outcomes. In order to assess whether this is the case we assess the Transformation activities defined for the process and investigate the work products that these activities produce. This provides the basis for assessing any business process.

PP-1 : The requirements for the process execution, the work products to be produced and the process outcomes to be achieved are defined.

PP-2 : Work products are produced which demonstrate that the process outcomes have been achieved.

PP-3 : Outcomes are achieved which demonstrate that the requirements have been met.

Questions :

A votre avis, la gestion des tests a-t-elle produit, d'une façon générale, les résultats attendus ?

Est-ce que tous les éléments, activités et résultats attendus définis par le processus existent ?

Une stratégie est-elle développée ?

Tous les éléments générés par le processus ou le projet sont-ils identifiés, définis et références ?

Performance Manager (PM)

The extent to which the performance of the process is managed to produce work products within defined time and resource requirements. Note : This attribute addresses whether the process is well planned i.e. what, when, who, how. It includes planning, tracking, measuring and closed loop process control of schedule and resource within the project in terms of measuring achievement against stated outcomes and taking corrective controlling action. It includes process risk management.

PM-1 : The resources required to perform the process are defined.

PM-2 : The timescale and/or cycle-time requirements for the performance of the process are defined.

PM-3 : The performance of the process is managed (planned, tracked and adjusted) to produce the work products and achieve the outcomes within the defined time and resource

requirements.

Questions :

Qui a travaillé à l'établissement et au suivi de la gestion des tests ? Est-ce que ces personnes étaient clairement identifiées comme ressources du processus ? Est-ce défini / formalisé quelque part ?

Avez-vous possibilité de pointer le temps passé sur les activités du processus ? (lien avec la " time sheet ")

Les tâches de gestion des tests est-elle planifiée ? Prévue explicitement dans le plan de projet ?

Le temps imparti à la gestion des tests est-il utilisé ?

Le planning a-t-il été suivi, revu, réajusté ? Le temps imparti à la gestion des tests était-il suffisant ? Les ressources allouées étaient-elles " suffisantes " ?

Le plan de gestion des tests a-t-il été suivi, revu, réajusté ?

Quality Control (QC)

The extent to which the performance of the process is managed to produce work products that meet their functional and non-functional requirements. Note : This attribute addresses the way the quality of process work products is managed. It includes the concept of closed loop process control of quality within the project in terms of measuring achievement against stated quality requirements and taking corrective action.

QC-1 : The work product functional and non-functional requirements are defined.

QC-2 : The quality of the work products is managed to ensure that they meet their functional and non-functional requirements.

Questions :

Existe t-il un guide pour mener à bien les activités de gestion des tests ?

S'il y a des documents relatifs à la gestion des tests, incluez-vous des exigences non fonctionnelles (terminologie, lisibilité, format ...) ?

Contrôlez-vous la qualité de la gestion des tests ? Comment ? (inclus dans une réunion qui existe déjà)

En particulier, en faites-vous des revues ?

Si lors de ces revues vous détectez des problèmes ou des erreurs, y a-il des actions correctives qui sont entreprises ?

Work Product Control (WPC)

The extent to which the performance of the process is managed to ensure that the work products are documented and controlled. Note : This attribute addresses whether the work products produced by the process are formally documented, secure and controlled, and whether changes and problems are correctly managed.

WPC-1 : The work product integrity requirements are defined.

WPC-2 : The work product dependencies are articulated.

WPC-3 : The configuration control, version control, access control and documentation of the work products are managed to ensure that they meet their integrity requirements.

Questions :

Est-ce que le processus de gestion des tests et les documents s'y rattachant sont eux-même gérés en test ?

y-a-t-il des règles de stockage des documents de gestion des tests ? Et une gestion des droits d'accès ?

L'outillage que vous utilisez pour la gestion des tests est-il géré en configuration ?

Process Definition (PD)

The extent to which the performance of the process uses a process definition based upon a standard process to achieve the defined process outcomes. Note : This attribute addresses whether the process is established across the organisation. This might be for the entire organisation or an organisational unit where there is significant process diversity across the entire organisation. It should include specific guidelines for tailoring the standard process to accommodate the goals of a specific instance.

PD-1 : Process documentation, together with appropriate guidance on tailoring of the standard process documentation, is defined which is capable of supporting the normal range of performance of the process and the work product functional and non-functional requirements.

PD-2 : The performance of the process is conducted in accordance with appropriately selected and/or tailored standard process documentation.

PD-3 : Historical process performance data is gathered to establish and refine the understanding of the process behavior in order to estimate the process performance resource needs.

PD-4 : Experiences of using the process documentation are used to refine the standard process.

Questions :

Y a t-il un guide décrivant la méthodologie à appliquer les activités de gestion des tests ?

Utilisez-vous le retour d'expériences sur ce processus, des informations issues du bilan de projet ?

Avez-vous un processus de capitalisation pour la gestion des tests après chaque projet ?

Comment répercutez-vous les changements occasionnés à la stratégie de gestion des tests ?

Technology Management (TM)

The extent to which the process draws upon a suitable process infrastructure that is appropriately allocated to deploy the defined process. Note :This attribute addresses whether a suitable environment and suitable technology is available to support the execution of the process.

TM-1 : The process infrastructure required for performing the process is documented.

TM-2 : The required amount of appropriate process infrastructure is available, allocated and used to support the performance of the process in line with the process documentation.

Questions :

Disposez-vous d'outils (au sens large) pour supporter la gestion des tests ?

Qu'est-ce que vous avez utilisé comme technologie pour mener à bien ce processus ?

Est-ce que cette technologie était suffisante ? Bien adaptée ?

Est-ce que votre environnement de travail vous a semblé adapté pour mener à bien ce processus ?

Est-ce qu'il y a une chose que vous regrettez de ne pas avoir eu à votre disposition (outil, matériel ...) ?

Est-ce que vous avez une documentation à disposition concernant les outils que vous avez utilisés pour ce processus ?

Human Resources (HR)

The extent to which the process draws upon competent human resources that are appropriately allocated to deploy the defined process. Note : This attribute addresses whether process deployment is being monitored across the established process and whether data is being collected on the performance of the process within specific instances. It provides a quantitative (statistical) understanding of the wider performance of the process.

HR-1 : The roles, responsibilities and competencies required for performing the process are documented.

HR-2 : The required amount of competent human resource is available, allocated and used to support the performance of the process in line with the process documentation.

Questions :

Les rôles et responsabilités en terme de gestion des tests étaient-ils définis au niveau organisationnel ?

Est-ce que vous pensez que le personnel est suffisamment formé pour mener à bien la gestion des tests ?

Y a-t-il eu des formations relatives à l'établissement de la gestion des tests ?

Process Control (PC)

The extent to which the process is controlled through the collection and analysis of product and process measures to correct, where necessary, the performance of the process to reliably achieve the defined process goals. Note : This attribute addresses whether process deployment is being monitored across the established process and whether data is being collected on the performance of the process within specific instances. It provides a quantitative (statistical) understanding of the wider performance of the process.

Pco-1 : Suitable analysis and control techniques are identified.

Pco-2 : In-process measures are collected and analysed.

Pco-3 : Process performance is managed to ensure it is within the defined limits.

Questions :

Au niveau du projet, est-ce qu'il y a des métriques qui ont été identifiées pour le processus ? De quelle manière (Technique Goal / Question / Metric par exemple ?)

De quelle manière les informations ont-elles été récoltées ? Analysées ? Exploitées, pour vérifier la performance du processus, et corriger des dysfonctionnements ?

A.2 Questionnaire sur les tests logiciels au CITI

Résumé

Ce document est un questionnaire type pour évaluer la façon dont les tests logiciels sont pratiqués au CITI.

Traitement des erreurs

Avez-vous une classification des erreurs ?

Si oui, de quelle forme ?

Classification par division ?

Organisez-vous les tests ?

EFFECTUEZ-VOUS UN PLANNING DE TESTS ? C'EST-A-DIRE ETABLISSEZ-VOUS LES OBJECTIFS, CRITERES ET STRATEGIE DES TESTS A EFFECTUER ?

Documentation des tests

Plan de test (qui décrit le système lui-même et le plan pour exercer toutes les fonctions et les caractéristiques) ?

Spécification et évaluation des tests (qui détaille tous les tests et définit le critère pour évaluer chaque fonctionnalité adressée par le test) ?

Description des tests (qui présente les données du test et procédures pour les tests individuels) ?

Analyse du rapport des tests (qui décrit les résultats de chaque test) ?

Utilisez-vous les rapports de tests ?

Quel est le cycle de vie des tests ? (Approche itérative ...)

Utilisez-vous des check listes ?

Les tests

QU'ELLES SONT LES TESTS EFFECTUES ? (ET DESCRIPTION DE CHAQUE TEST)
TYPES DE TESTS :

Les tests unitaires
Les tests d'intégration
Les tests fonctionnels
Les tests de non-régression
Les tests de performances
Les tests d'acceptation
Les tests IHM
Les tests de surcharge
Les tests de volume
Les tests de configuration

Les tests de compatibilité
Les tests de sécurité
Les tests de séquençage
Les tests environnementaux
Les tests de reprise
Les tests de documentation
Les tests des facteurs humains
Les tests système
Les tests de robustesse
Les test d'installation
Les tests de parallélisme

UTILISEZ-VOUS DES OUTILS DE TESTS AUTOMATIQUES ? (POUR QUELS TESTS)

Méthode et modèle de développement de logiciel

SUIVEZ-VOUS UNE METHODE OU UN MODELE POUR EFFECTUER LE DEVELOPPEMENT DU LOGICIEL ? (XP, RAD, RUP, CMM, SPICE ...) SI METHODE INTERNE DEMANDER DOCUMENTATION.

Ressource humaine

FORMEZ-VOUS UNE EQUIPE DE TEST DES LE DEBUT DU DEVELOPPEMENT OU DES PERSONNES SONT-ELLES ASSIGNES A UNE TACHE ? (EQUIPE OU UN TESTEUR, UN CONCEPTEURĚ) + DESCRIPTION.

Il y a-t-il une personne connaissant le domaine traité ?

Intervention du client/utilisateur

QUAND ET OU LE CLIENT/UTILISATEUR INTERVIENT-IL DANS LES TESTS ?

A.3 Rapport d'évaluation : Gestion des tests logiciels au CITI

[1]

Résumé

Ce document présente les résultats de l'évaluation concernant les pratiques des tests logiciels au CITI.

Introduction

Cette évaluation concerne les pratiques des tests logiciels au CITI, afin de faire un état des lieux de ces pratiques, et d'initialiser un programme d'amélioration du processus des tests logiciels.

Contexte et objectifs de l'évaluation

Objectif de l'évaluation :

- déterminer le niveau de maîtrise du processus de test logiciels (profil de maturité), en tenant compte des objectifs stratégiques du CITI ;
- identifier les forces et faiblesses, les risques et les opportunités d'amélioration dégagés à partir de l'évaluation réalisée ;
- recommander des mesures précises permettant d'élaborer un programme d'améliorations progressives à mettre en place.

Portée :
L'évaluation porte sur la gestion des tests logiciels au sein du CITI, et ne s'intéresse pas à d'autres processus.

Dates clés :

- Début des interviews : 28 octobre 2003.
- Fin des interviews : 3 novembre 2003.

L'évaluateur :
Michaël S'Jongers, stagiaire, unité REF.

Méthode :
L'évaluation a été menée en réalisant une série d'entretiens individuels avec des collaborateurs du CITI ayant participé à des projets de développement informatique.

La méthode

Cette section décrit la méthode appliquée pour évaluer les processus. Le lecteur trouvera les informations de base pour interpréter le profil résultat.

Le modèle

La méthode utilisée repose sur la norme SPICE (ISO 15504). Elle propose un modèle composé de processus typiquement appliqués dans le domaine de l'ingénierie informatique. Ces processus sont classés en 5 catégories (Acquisition et Fourniture, Ingénierie, Support,

[1] Le CITI n'a pas permis la diffusion de l'évaluation.

Management, Organisation) selon leur domaine d'application. Pour chaque processus, le modèle décrit sa définition, ses objectifs, ses activités de transformation (c'est à dire des pratiques de base) ainsi que les produits entrants et sortants. Le modèle définit des attributs génériques, communs à tous les processus, permettant d'établir un profil par processus évalué, structuré de la même façon.

L'échelle d'évaluation

Chaque attribut est évalué sur une échelle à quatre niveaux. Cette échelle reflète à quel point l'objectif associé à l'attribut a été atteint en pratique.

Complètement atteint	objectif atteint à 100%	(FULLY)
Largement atteint	objectif atteint à 70%	(LARGELY)
Partiellement atteint	objectif atteint à 30%	(PARTIALLY)
Pas atteint	objectif pas du tout atteint	(NOT)

TAB. A.1 – Echelle d'évaluation des objectifs

Le profil

La représentation graphique des résultats d'évaluation des attributs est appelée profil de capacité actuel ou profil de maturité. Ce profil reprend horizontalement les attributs et verticalement les processus évalués. Cette représentation permet de dégager de manière visuelle les écarts communs aux processus ou les écarts induits et croisés.

Exemple de profil d'évaluation

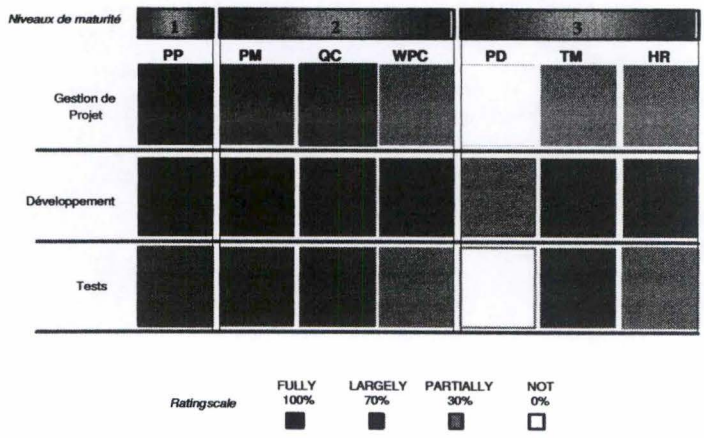


FIG. A.1 – Profile d'évaluation

Les niveau de maturité

Le profil donne une bonne vue d'ensemble sur les pratiques en cours des processus évalués. La méthode propose une façon plus abstraite pour classer les profils en niveaux de maturité. La cible est basée sur une matrice qui associe à chaque processus le niveau que les différents attributs doivent atteindre. Les niveaux de maturité sont décrits dans le tableau qui suit.

Niveau	Définition
Niveau 0 - Incomplet (Initial)	Le processus n'est pas réalisé ou n'atteint que partiellement les objectifs définis. (The process is not implemented or fails to achieve its defined goals.)
Niveau 1 - Réalisé (Performed)	Le processus implémenté atteint ses objectifs définis. (The implemented process achieves its defined goals.)
Niveau 2 - Géré (Managed)	Le processus exécuté délivre des résultats selon une qualité définie, dans les délais et selon les ressources préalablement définies.(The Performed process delivers work products of defined quality within defined timescales and resource needs.)
Niveau 3 - Etabli (Established)	Le processus géré est exécuté selon la définition du processus et en utilisant les ressources qui conviennent.(The Managed process is performed using a defined process and utilising suitable resources.)
Niveau 4 - Prévisible (Predictable)	Le processus établi est exécuté selon des limites définies et contrôlées, en correspondance avec les objectifs stratégiques de l'organisation.(The Established process is performed within defined control limits in line with the business goals of the organisation.)
Niveau 5 - En optimisation (Optimising)	Le processus maîtrisé quantitativement est optimisé afin de satisfaire les objectifs stratégiques actuels et futurs, de l'entreprise.(The Predictable process is optimised to meet current and future business goals.)

TAB. A.2 – Echelle d'évaluation de la maturité

Chaque niveau se construit sur le précédent et se décompose en un ou plusieurs attributs qui le caractérisent. La description de l'association des niveaux des attributs est donnée en annexe. Les résultats classifiés en niveaux peuvent typiquement être utilisés pour être distribués dans des bases de données de benchmarking. Si une cible est déterminée pour les processus de l'évaluation des pratiques logicielles avec un niveau 3, ceci correspond à la cible suivante par rapport aux attributs :

Process Performance (PP) : Fully (Couverture et atteinte des objectifs du processus)

Performance Management (PM) : Fully

(Gestion et suivi des activités du processus)

Quality Control (QC) : Fully

(Contrôle de la qualité)

Work Product Control (WPC) : Fully

(Contrôle des résultats)

Process Definition (PD) : Largely or Fully

(Définition du processus)

Technology Management (TM) : Largely or Fully

(Gestion de la technologie)

Human Resource Management (HR) : Largely or Fully

(Gestion des Ressources Humaines)

Date	Heure	Personne interrogée	Projet(s)

Entrée SPICE	Eléments recueillis	Sortie SPICE	Eléments recueillis

Résultats de l'évaluation du processus de test logiciel

Liste des personnes interrogées

Recensement des éléments en entrée et en sortie du processus

Profil du processus

Impréssions générales sur l'interprétation du profil

PB1	Développer une stratégie d'intégration et de test du système. Développer une stratégie d'intégration des paquetages/modules de système et de test système cohérent avec la stratégie de livraison.
PB2	Développer une stratégie de test de non-régression. Développer une stratégie de ré-exécution des tests des composants/éléments si un changement a été introduit dans un paquetage/module de système.
PB3	Construire les paquetages/modules du système. Identifier les paquetages du système et une séquence ou un ordre pour les tests.
PB4	Développer les tests pour les paquetages/modules du système. Décrire les tests à réaliser pour chaque paquetage/module du système, en indiquant les exigences à contrôler, les données en entrée, les composants/éléments de système nécessaires pour réaliser les tests et les critères d'acceptation.
PB5	Vérifier les paquetages/modules du système. Vérifier chaque paquetage/module du système pour s'assurer qu'il satisfait aux exigences, et documenter les résultats.
PB6	Développer les tests pour le système. Décrire les tests à réaliser pour le système intégré, en indiquant les exigences du système à contrôler, les données en entrée et les critères d'acceptation.
PB7	Vérifier le système intégré. Vérifier le système intégré pour s'assurer qu'il satisfait aux exigences du système, et documenter les résultats.
PB8	Vérifier la non-régression des paquetages/modules du système ou le système intégré. Si des changements sont apportés dans les composants/éléments du système, exécuter les tests de non-régression tels que définis dans la stratégie de test de non-régression.

Gestion et suivi des activités du Processus (Process Performance - PP)

-

Gestion et suivi des activités du Processus (Performance Management - PM)

-

Contrôle de la qualité et contrôle des résultats (Quality Control - QC et Work Product Control - WPC)

-

Définition du processus (Process Definition - PD)

-

Gestion de la Technologie et des Ressources Humaines (Technology Management - TM et Human Resource Management - HR)

-

Analyse SWOR(STRENGTHS/WEAKNESSES/OPPORTUNITIES/RISKS)

Forces	
Risque	
Faiblesses	
Opportunités	

Analyse des entretiens

-

Conclusion

-

Annexe B

Modèle de document

B.1 Modèle de plan de test

Résumé

Ce document contient un modèle de plan de test.

Introduction

But

Ce plan de test pour le projet *[nom du projet]* a les objectifs suivants :

- *[Identifier toutes les informations disponibles sur le projet et sur les composants du logiciel devant être testés ;*
- *Faire la liste de toutes les exigences recommandées pour les tests ;*
- *Décrire la stratégie de test qui va être développée ;*
- *Identifier les ressources nécessaires pour les tests.]*

Portée

- *[Décrire les étapes et les types de test qui sont concernés par ce plan de test.*
- *Produire une liste des fonctionnalités qui doivent ou ne doivent pas être testées.]*

Documents

La table ci-dessous indique les documents disponibles pour l'activité de test.

- *[Remplir le tableau avec les documents disponibles ;*
- *Marquer d'une croix les documents disponibles ;*
- *Rajouter les documents non listés.]*

Document	Version	Disponible	Notes
Cas d'utilisation			
Documents d'architecture			
Planning du projet			
Spécifications supplémentaires			

TAB. B.1 – Tableau des documents disponibles

Les exigences des tests

[Faire la liste de tous les éléments qui doivent passer les tests (fonctionnel et non fonctionnel).]

La stratégie de test

- *[La stratégie de test présente l'approche recommandée, les techniques et les outils utilisés pour les tests ;*
- *Pour chaque type de test, produire une description du test et pourquoi il est exécuté ;*
- *Signaler tous les tests qui ne seront pas effectués. Mettre une phrase de justification comme par exemple en signalant que ce test n'est pas important à ce niveau ;*
- *Indiquer la technique utilisée pour chaque type de test, c'est-à-dire décrire comment les tests sont exécutés ;*
- *Indiquer les critères d'achèvement des tests, c'est-à-dire indiquer les valeurs qui déterminent la fin des tests, en indiquant ce qui doit être testé, comment les mesures sont prises et le critère qui est utilisé pour évaluer les mesures.]*

Les types de test

Les tests unitaires

Objectif du test	
Technique	
Critère d'achèvement	
Considération spéciale	
Responsable	

TAB. B.2 – Tableau de description des tests unitaires

Les tests système

Objectif du test	
Technique	
Critère d'achèvement	
Considération spéciale	
Responsable	

TAB. B.3 – Tableau de description des tests système

Les tests d'acceptation

Objectif du test	
Technique	
Critère d'achèvement	
Considération spéciale	
Responsable	

TAB. B.4 – Tableau de description des tests d'acceptation

Les outils

[Faire une liste de tous les outils utilisés dans le projet pour réaliser les tests.]

Outil	Version	Commentaire
...	...	...

TAB. B.5 – Tableau des documents disponibles

Les ressources

[Cette section présente les ressources recommandées pour le projet.]

Les ressources humaines

[Remplir le tableau en indiquant : La personne, son rôle et ses responsabilités.]

Personne	Rôle(s)	Responsabilité(s)
...	...	...

TAB. B.6 – Tableau des ressorces humaines

Les ressources systèmes

[Remplir le tableau suivant avec les ressources système dont les tests auront besoin. Par exemple les bases de données, les serveurs, les configurations spéciales ...]

Ressource	Nom/type
...	...

TAB. B.7 – Tableau des ressorces systèmes

Le planning des tests

Le planning

[Insérer un planning ou mettre une référence vers un fichier de planification.]

Les jalons

[Il faut identifier les jalons (bornes) des tests pour mieux communiquer entre les activités et avoir une idée sur le début et la fin de chaque activité.]

Activité	Charge de travail	Début	Fin
...	...	...	...

TAB. B.8 – Tableau des jalons

Les livrables de l'activité de test

[Faire la liste de tous les documents, les rapports créés en indiquant pour qui et quand.]

Délivrable	Destination	Origine	Date
...	...	...	...

TAB. B.9 – Tableau des livrables

B.2 Modèle pour les cas de test

Résumé

Ce document est un modèle pour les cas de test. Ce document peut être élaboré à l'aide du guide de création des cas de test.

Introduction

Nom du rédacteur	
Version de l'application	
Fonctionnalité cible	
Date des tests	

TAB. B.10 – Description du cas de test

Scénarii

Nom :

Description :

Pré-condition :

Post-condition :

Scénario de base (flux de base) :

Actions utilisateur	**Actions système**
...	...

TAB. B.11 – Scénario de base

Scénarii alternatifs (flux alternatifs) :

[Pour chaque flux alternatif : Nom, description, pré/post-condition et variations par rapport au flux de base.]

Cas de test

Matrice des cas de test

Cas de test	**Scénario**	**Conditions**	**Résultats attendus**
...	...	...	...

TAB. B.12 – Matrice des cas de test

Résultat des tests

Cas de test	OK/KO	Commentaire
...	...	...

TAB. B.13 – Tableau des livrables

B.3 Récapitulatif des cas de test

Résumé

Ce document est un modèle pour le récapitulatif des cas de test pour l'ensemble du projet.

Matrice des cas de test

Paquetage/classe	Cas de test	Scénario	Con onditions	Résultats attendus
...	...		...	...

TAB. B.14 – Matrice des cas de test

Résultats des tests

Paquetage/classe	Cas de test	OK/KO	Commentaire
...	...	...	...

TAB. B.15 – Résultat des tests

B.4 Modèle de fiche de livraison

[Nom du projet]

Description du livrable

Libellé :

Version packagée :

Date :

Résumé

Ce document est un modèle de fiche de livraison.

Procédure d'installation

Consignes d'installation

[Décrire les consignes d'installation : par exemple l'adresse du fichier d'installation et une description des étapes d'installation.]

Authentification

[Décrire la procédure d'authentification des testeurs (login et mot de passe) et décrire les droits accordés.]

Description de la version livrée

Fonctionnalités couvertes par la version livrée

[Indiquer ici toutes les fonctionnalités couvertes par la version livrée.]

Fonctionnalités nouvelles

[Indiquer ici les fonctionnalités nouvelles par rapport à la version précédente.]

Tests prioritaires

[Indiquer ici les tests à réaliser de façon prioritaire.]

Anomalies corrigées dans cette version

[Indiquer ici les références des anomalies corrigées dans cette version.]

Anomalies non corrigées dans cette version

[Indiquer ici les références des anomalies connues mais non encore corrigées. La phrase type suivante peut être suffisante. Toutes les anomalies ouvertes qui ne font pas partie de la section précédente doivent être considérées comme n'étant pas corrigées par cette version. Il n'est donc pas nécessaire de les tester ou de saisir de nouvelles anomalies pour attester que l'erreur subsiste .]

Fonctionnalités non disponibles - A ne pas tester

[Indiquer les fonctionnalités qui sont accessibles par le testeur mais qui ne sont pas finies et qui ne doivent pas être testées.]

Répertorier les anomalies

Afin de répertorier au mieux les tests effectués et les différents bugs ou dysfonctionnement repérés, il est demandé au testeur :

[Indiquer la procédure de suivi de la réalisation des tests Indiquer la procédure de saisie des demandes d'évolution et des fiches d'anomalies .]

Annexe C

Check listes

C.1 Check liste pour le plan de test

Résumé

Ce document est une check liste pour le plan de test.

Check liste

OK/KO	Question
	Le plan de test identifie-t-il clairement la portée des tests ?
	Le plan de test identifie-t-il clairement les étapes de test et les types de test ?
	Le plan de test identifie-t-il clairement les fonctions qui devront être ou non testées ?
	Le plan de test identifie-t-il clairement les ressources humaines pour exécuter les tests ?
	Le plan de test identifie-t-il clairement les documents utiles aux tests ?
	Le plan de test identifie-t-il clairement les outils utiles aux tests ?
	Toutes les ressources (matérielles, logicielles et humaines) pour les tests sont-elles identifiées et disponibles ?
	Une stratégie de test est-elle identifiée dans le plan de test ?
	Y a-t-il un objectif dans la stratégie ?
	Y a-t-il une description sur comment tester ?
	Y a-t-il un critère d'achèvement des tests ?
	Y a-t-il des jalons définis dans le plan de test ?
	Les livrables sont-ils identifiés ?
	Le planning des tests est-il présenté dans le plan de test ?

C.2 Check liste pour les cas de test

Résumé

Ce document est une check liste pour les cas de test.

Check liste

OK/KO	Question
	Les scénarii ont-ils un nom, une description, et une pré et post-condition ?
	Pour chaque scénario, il y a-t-il au moins deux cas de test : un cas positif (condition normal) et un cas négatif (condition anormal) ?
	Les cas de test redondants sont-ils tous éliminés ?
	Les cas de test ont-ils tous un identifiant, une condition, un résultat attendus et des données d'entrée ?
	Les cas des tests ont-ils une date prévue pour leur exécution ?

Annexe D

Guides

D.1 Création des cas de test

Résumé

Ce document constitue un guide de création des cas de test, à partir des cas d'utilisation pour les tests fonctionnels et à partir des exigences non-fonctionnelles pour les tests de performance.

Qu'est-ce qu'un cas de test ?

Les cas de test sont des « *Instructions écrites à l'usage de l'exécutant des tests qui spécifient la manière dont une fonction ou une combinaison de fonctions doit être testée.*

Un cas de test contient des informations détaillées sur les points suivants :

- *L'objectif du test* ;
- *Les scénarios de test* ;
- *Les tests élémentaires* ;
- *Les procédures de test.* » [ISO02]

C'est un ensemble d'entrées de test, de conditions d'exécution et de résultats attendus développés pour un objectif particulier, tel qu'exercer un programme particulier ou pour vérifier la conformité avec un besoin spécifique. L'ensemble des cas de test doit rencontrer les besoins exigés. Pour cela, il suffit de définir des cas de test pour chaque fonctionnalité que le système doit satisfaire.

Cas de test pour les tests fonctionnels

Les tests fonctionnels se basent sur des cas de test construits à partir des cas d'utilisation. Les cas d'utilisation sont l'expression de l'ensemble des fonctionnalités du système.

La création d'un cas de test dérivé à partir d'un cas d'utilisation se fait en quatre étapes :

- Identifier tous les scénarii d'un cas d'utilisation ;
- Identifier un ou plusieurs cas de test pour chaque scénario ;
- Pour chaque cas de test, identifier les conditions d'exécution dans le cas d'utilisation ;
- Eliminer les cas redondants ;
- Ajouter les valeurs des données pour les conditions du cas de test.

Identification des scénarii d'un cas d'utilisation

La technique

Les scénarii possibles sont identifiés par le flux de base et les flux alternatifs du cas d'utilisation. Le flux de base est le déroulement normal d'un cas d'utilisation et les flux alternatifs sont des déroulements alternatifs (cas de problèmes, erreurs, utilisation moins habituelle ...).

Pour trouver tous les scénarii possibles d'un cas d'utilisation, il faut :

- Identifier le flux de base du cas d'utilisation c'est-à-dire le déroulement normal du cas d'utilisation,
- Identifier tous les flux alternatifs du cas d'utilisation c'est-à-dire tous les déroulements alternatifs à partir du déroulement normal,
- Chaque flux représente un scénario.

L'exemple

Exemple d'un système d'achat de billet d'avion.

Soit une borne qui permet d'acheter des billets d'avions. Un opérateur peut venir approvisionner la borne en billets. On distingue plusieurs types de client. Le client normal qui peut acheter un billet en payant comptant, le client dit « Frequent Traveller » qui peut acheter un billet en payant comptant, obtenir des points et consulter les points, et le client « Premium » qui est un client « Frequent Traveller » mais qui peut payer le billet en payant en différé. Le système offre différents moyens de paiement : le paiement par carte de crédit via le réseau Bankcard, le paiement par carte de débit via le réseau Banksys, et le paiement par carte Proton via le réseau Proton.

Le diagramme de cas d'utilisation est :
On prend pour exemple la fonctionnalité « obtenir point » :
Le cas d'utilisation obtenir point :

Nom : Obtenir point Brève

description : Un client Frequent Traveller fait comptabiliser les points auxquels son titre de voyage donne droit.

Pré-condition du cas normal :
- Le client est présent à la borne
- La borne fonctionne correctement
- La borne est prête à accueillir un nouveau client

Post-condition du cas normal :
- Les points du client ont été comptabilisés
- La borne fonctionne correctement
- La borne est prête à accueillir un nouveau client

Le flux de base (scénario de base) :
Les flux alternatifs (scénarii alternatifs) :

Qu'est-ce qui peut se passer différemment ?

1. Le client est déjà identifié car il vient d'effectuer d'autres opérations ;

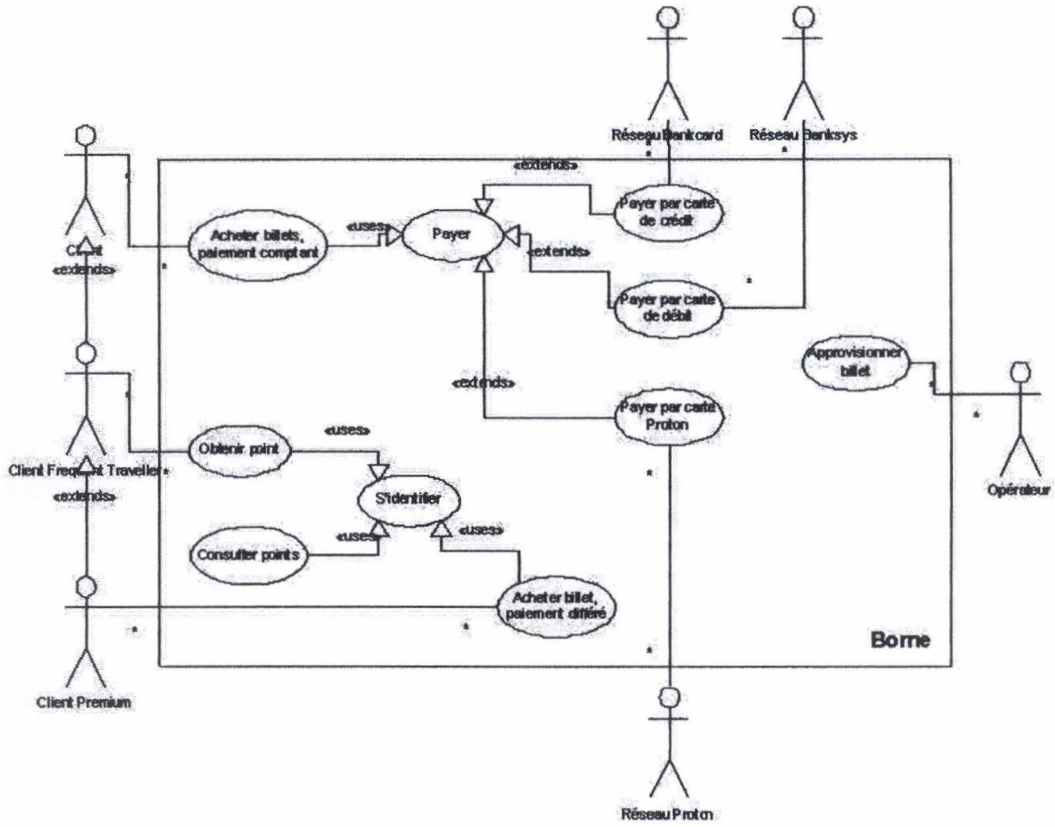


FIG. D.1 – Diagramme de cas d'utilisation de la borne

2. Le client désire effectuer d'autres opérations après celle-ci ;
3. Le client désire faire comptabiliser plusieurs billets ;
4. La carte est insérée mais n'a pas le format attendu ;
5. Le code ne correspond pas à la carte ;
6. Le code n'est pas introduit en moins de 30 secondes ;
7. La piste magnétique de la carte est corrompue ;
8. La date d'expiration de la carte est passée ;
9. Le billet n'a pas le format attendu ;
10. Le billet ne rentre pas dans la fente ;
11. Le client repart en oubliant le billet.

Flux alternatif nř1 : Client déjà identifié

Nom : Obtenir point (client déjà identifié)
Pré-condition :
Idem scénario normal
Le client est déjà identifié
Post-condition :
Idem cas normal
Scénario du cas alternatif nř1 :
Si le client est déjà identifié, les interactions commencent directement au point 3 du scénario de base.

Flux alternatif nř2 :

Client	Borne
1. Le client insère sa carte de Frequent Traveller et compose son code	
	2. Le système s'assure que la carte est valide et que le code est correct
3. Le client signale son intention de faire comptabiliser des points	
	4. Le système demande au client d'insérer son billet usagé
5. Le client insère son billet	
	6. Le système s'assure que le billet est valide
	7. Le système comptabilise les points
	8. Le système rend le billet au client
9. Le client reprend le billet et signale qu'il a terminé	
	10. Le système rend la carte au client
11. Le client reprend sa carte	
	12. Le système est à nouveau prêt à accueillir un client

Client désire effectuer d'autres opérations

Nom : Obtenir point (client désire effectuer d'autres opérations)

Pré-condition :
Idem scénario normal

Post-condition :
Les points du client ont été comptabilisés
La borne fonctionne correctement
La borne attend la demande du client d'une nouvelle opération

Scénario du cas alternatif nř2 :
Si le client désire effectuer d'autres opérations après avoir obtenu des points (après le point 8) alors le scénario se termine au point 9.

Ainsi de suite ...

Identification des cas de test et leurs conditions d'exécution

La technique

On utilise les scénarii identifiés à l'étape précédente pour déterminer les cas de test. Un document de description de cas de test est élaboré pour chaque cas d'utilisation. Il y a un scénario pour chaque flux d'un cas d'utilisation et un ou plusieurs cas de test pour chaque scénario en fonction des conditions d'exécution.

Pour dériver les cas de test des scénarii, il faut d'abord identifier la condition spécifique qui cause l'exécution d'un scénario spécifique. C'est à dire qu'il faut identifier la condition qui provoque l'exécution d'un scénario. Ensuite, il faut identifier, pour chaque scénario, au moins un cas de test. Enfin, il suffit de créer une matrice des cas de test.

La matrice de cas de test se présente comme cela :

Cas de test	Scénario	Conditions	Résultats attendus
C1	SC1	...	...
Cn	SCn	...	...

Cette matrice permet de voir si les cas de test sont suffisants et si les conditions sont testées. Il doit y avoir des cas de test positif et négatif. C'est à dire qu'il faut aussi mettre des cas où l'exécution se termine bien.

L'exemple

Matrice des cas de test de l'exemple du système d'achat de billet d'avion pour la fonction obtenir point :

Cas de test	Scénario	Conditions	Résultats attendus
C01	SC01	Scénario de base avec un client « Frequent Traveller »	Les points sont comptabilisés
C02	SC01	Scénario de base avec un client « Premium »	Les points sont comptabilisés
C03	SC02	Le client est déjà identifié dans le système	Les points sont comptabilisés
C04	SC03	Le client « Frequent Traveller » veut effectuer une autre opération après	Le système est prêt pour une autre opération
...	...	...	...

Identification des cas redondants

La technique

La technique est d'éliminer le cas qui est déjà couvert par un autre cas. Un cas est couvert par un autre lorsque la ou les conditions d'exécution sont déjà comprises dans un autre cas et que le résultat attendu est le même. Il faut alors éliminer le cas qui est dit redondant.

L'élimination des cas redondants évite de tester deux fois la même chose, et d'avoir une longue liste de cas de test.

Ajouter les valeurs des données

La technique

Une fois que la matrice est créée, il suffit d'identifier les données qui rentrent dans les conditions des cas de test possibles.

Cas de test pour les tests de performance

La technique

Les cas de test pour les tests de performance sont dérivés des spécifications supplémentaires, c'est-à-dire des exigences non-fonctionnelles.

Il faut procéder comme-ceci :

- Identifier au moins un cas de test pour chaque critère de performance. Les critères de performance sont généralement exprimés en temps par transaction et en nombre de transaction,
- Identifier au moins un cas de test pour chaque critère de volume ou de charge du système.
- Identifier les conditions spécifiques qui affectent le temps de réponse comme la taille des données, le nombre d'utilisateurs simultanés, le nombre et le type de transactions simultanées qui sont accomplies, et les caractéristiques de l'environnement comme le matériel et le logiciel.

L'exemple

Exemple de test de charge pour les distributeurs de billet de banque :

Cas de test	Charge de travail	Conditions	Résultats attendus
C1	Un seul distributeur	La transaction de retrait est complète	Transaction complète en moins de 20 secondes
C2	1.000 distributeurs simultanément	La transaction de retrait est complète	Transaction complète en moins de 30 secondes
C3	10.000 distributeurs simultanément	La transaction de retrait est complète	Transaction complète en moins de 50 secondes

Cas de test pour les tests de non-régression

Les tests de non-régression s'assurent que toute nouvelle version d'un logiciel a au moins le même niveau de qualité que la version précédente. Ils vérifient que les nouveaux éléments ou les corrections d'anomalies introduits dans la nouvelle version ne remettent pas en cause la qualité de ce qui existait déjà. Pour s'en assurer, il faut refaire les tests effectués sur la version précédente et vérifier que de nouvelles anomalies n'apparaissent pas. Les cas de test pour les tests de non-régression sont donc les cas de test de l'itération précédente.

D.2 Description de demande d'évolution

Résumé

Ce document présente les données décrivant une demande d'évolution.

Description de la demande d'évolution

Introduction

Ce document présente les données utiles pour remplir une demande d'évolution complète pour une anomalie ou une modification. La section sur la demande d'évolution pour une modification indique les champs supplémentaires ou les champs qui n'ont pas d'intérêt par rapport à la demande d'évolution pour une anomalie.

Demande d'évolution pour une anomalie

Le signalement de l'anomalie

⊙ Le projet où l'anomalie est trouvée ;
⊙ La version du projet ;
⊙ La personne qui signale l'anomalie ;
⊙ La localisation de l'anomalie ;
⊙ La nature de l'anomalie (IHM, BD, applicative) ;
⊙ La date d'entrée de l'anomalie ;
⊙ La reproductibilité de l'anomalie (quelques fois, aléatoire, n'a pas essayé, toujours, impossible à reproduire et N/A) ;
⊙ La sévérité de l'anomalie (simple, texte, fonctionnalité, cosmétique, mineur, majeur, crash, bloquant) ;
⊙ La priorité de l'anomalie pour celui qui la rapporte (aucune, basse, normale, élevée, urgente, immédiate, ou par des chiffres) ;
⊙ Un résumé de la description ;
⊙ La description de l'anomalie ;
⊙ Les étapes aboutissant à la survenue d'une anomalie ;
⊙ Le contexte de survenue de l'anomalie ;
⊙ Des informations complémentaires à la description ;
⊙ Des captures d'écrans ;
⊙ La plate-forme (Windows, MAC, Unix ou Linux) ;
⊙ Le navigateur (Explorer, Netscape, Notes, autres navigateurs).

L'assignation d'une anomalie

⊙ Le responsable de la correction de l'anomalie ;
⊙ La priorité de l'anomalie ;
⊙ La description de l'impact que pourrait avoir la correction.

Le refus de la demande

⊙ La justification du refus de la demande.

La revue de la demande avant correction

⊙ Le temps de correction ;
⊙ Le descriptif de la correction ;

⊙ Le nom du correcteur ;
⊙ La version incluant la correction ;
⊙ La date de correction.

La validation de la correction

⊙ La personne fermant la demande ;
⊙ La date de fermeture ;
⊙ La justification /le commentaire de fermeture.

Toute action

⊙ Commentaires ajoutés sur l'anomalie ;
⊙ Un historique de l'anomalie (date d'entrée, date de dernière modification, les champs qui ont été modifiés ...).

Demande d'évolution pour une modification

Demande de modification

Champs supplémentaires par rapport au anomalie

⊙ La personne source de la demande ;
⊙ La justification de la modification ;
⊙ L'historique de la décision.

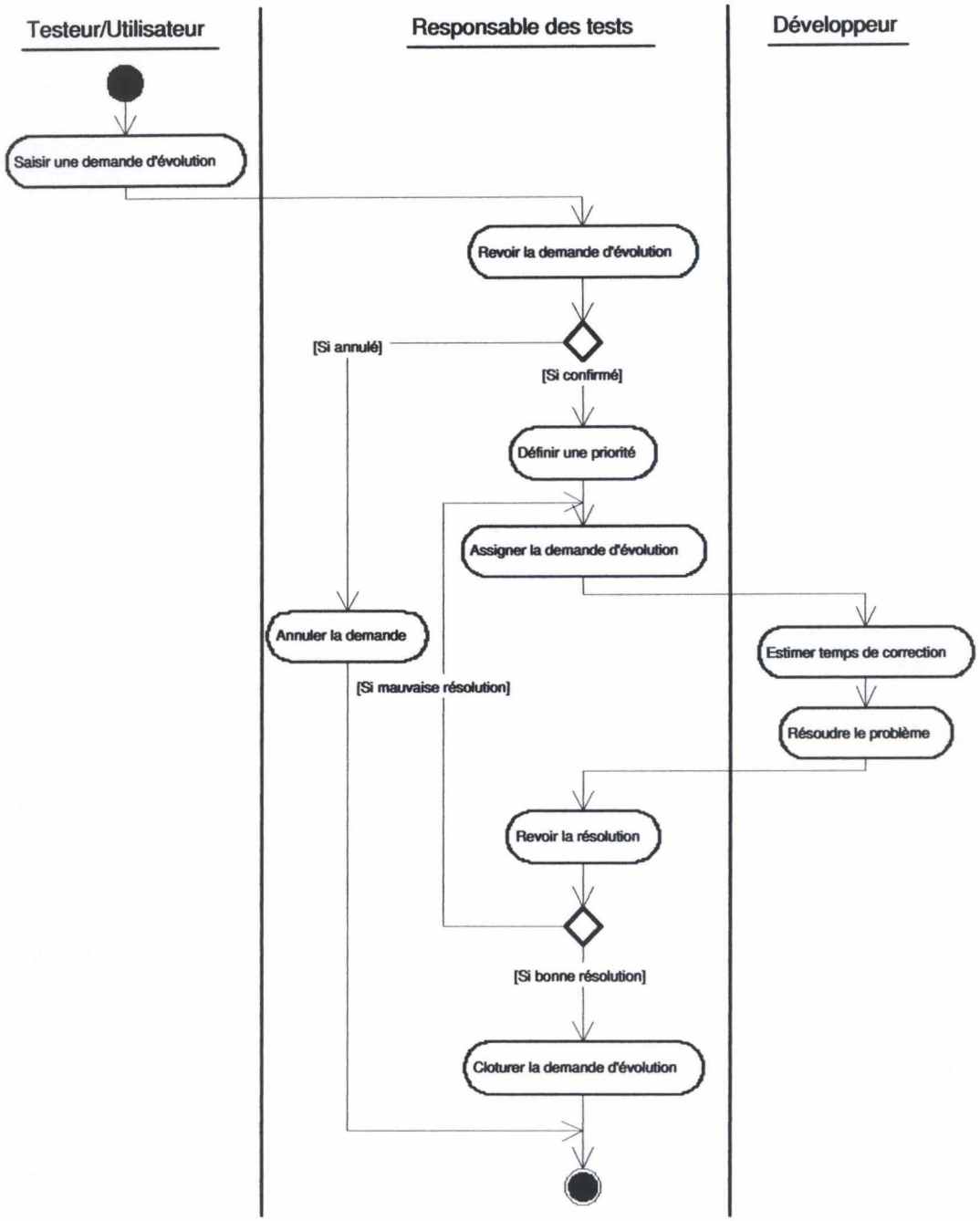
Champs sans intérêts

⊙ La reproductibilité d'une modification ;
⊙ La sévérité d'une modification.
⊙ La plate-forme (Windows, MAC, Unix ou Linux) ;
⊙ Le navigateur (Explorer, Netscape, Notes, autres navigateurs).

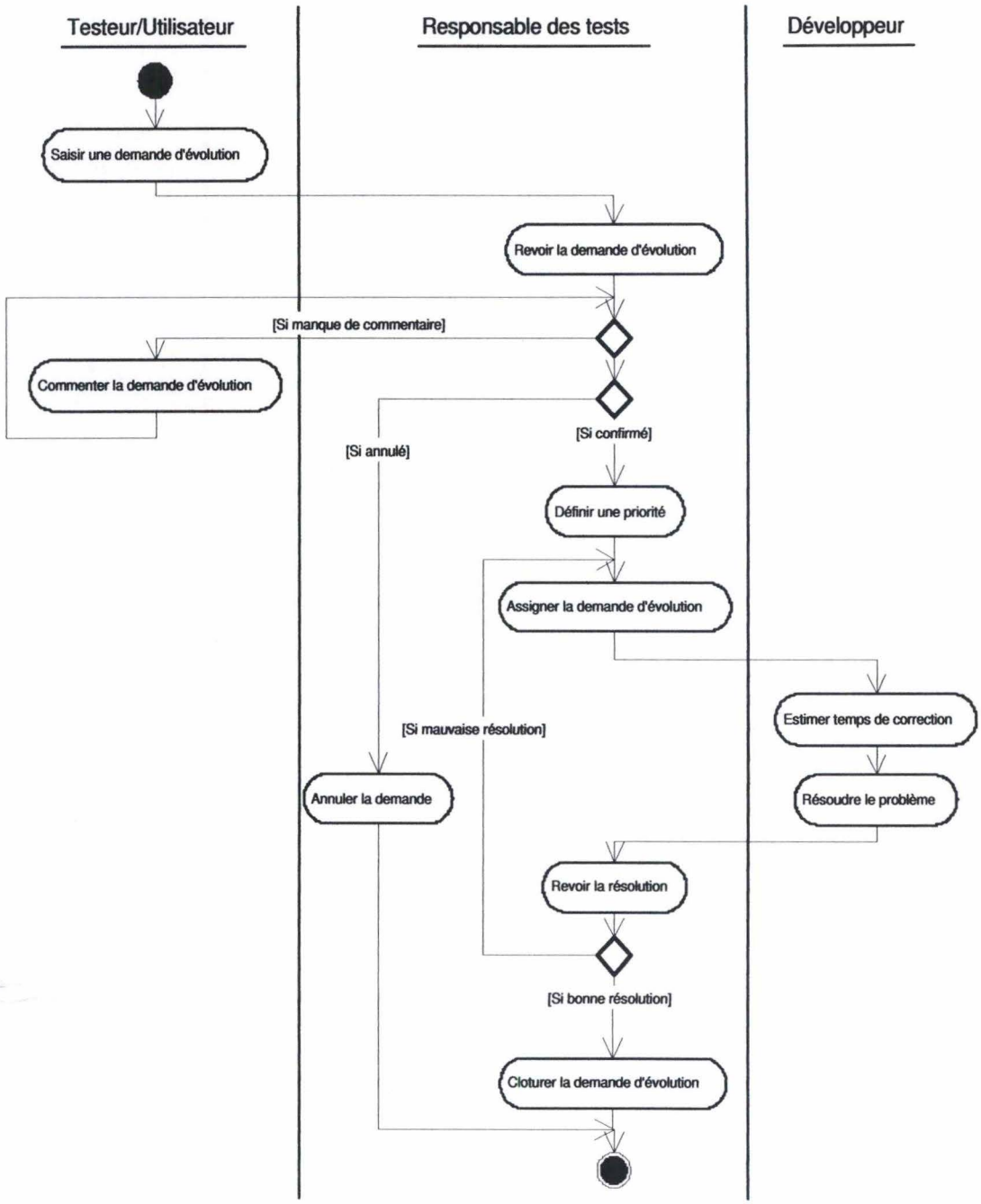
NB : Les autres points sont les mêmes que la demande d'évolution pour une anomalie.

Les diagrammes d'activité de gestion des demandes d'évolution

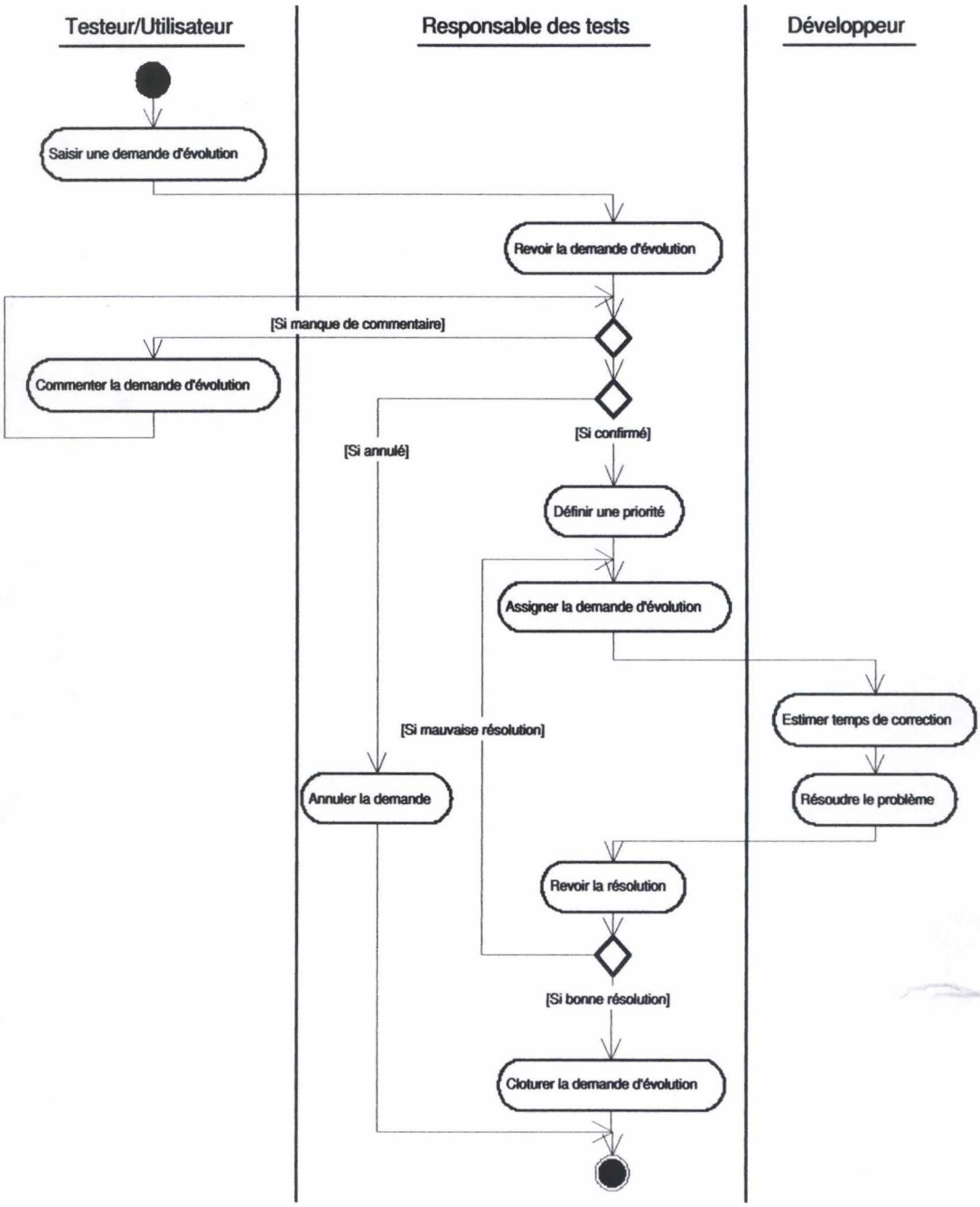
La Base Notes « Rapports de tests »



Mantis


Testeur/Utilisateur
Responsable des tests
Développeur
Saisir une demande d'évolution
Revoir la demande d'évolution
[Si manque de commentaire]
Commenter la demande d'évolution
[Si annulé]
[Si confirmé]
Définir une priorité
Assigner la demande d'évolution
Estimer temps de correction
[Si mauvaise résolution]
Annuler la demande
Résoudre le problème
Revoir la résolution
[Si bonne résolution]
Cloturer la demande d'évolution

Flux idéal


Testeur/Utilisateur
Responsable des tests
Développeur
Saisir une demande d'évolution
Revoir la demande d'évolution
[Si manque de commentaire]
Commenter la demande d'évolution
[Si annulé]
[Si confirmé]
Définir une priorité
Assigner la demande d'évolution
Estimer temps de correction
[Si mauvaise résolution]
Annuler la demande
Résoudre le problème
Revoir la résolution
[Si bonne résolution]
Cloturer la demande d'évolution